1978

Microcomputer based flight data recorder/monitor with solid state memory.

Easton, Darl Eugene

MICROCOMPUTER BASED FLIGHT DATA RECORDER/MONITOR WITH SOLID STATE MEMORY.

Darl Eugene Easton

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

MICROCOMPUTER BASED FLIGHT DATA
RECORDER/MONITOR WITH SOLID STATE MEMORY

by

Darl Eugene Easton

June 1978

Thesis Advisor: Uno. R. Kodres

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)
Microcomputer Based Flight Data Recorder/Monitor with Solid State Memory

5. TYPE OF REPORT & PERIOD COVERED
Master's Thesis; June 1978

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
Darl Eugene Easton

8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Naval Postgraduate School
Monterey, California 93940

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS
Naval Postgraduate School
Monterey, California 93940

12. REPORT DATE
June 1978

13. NUMBER OF PAGES
144

14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)
Naval Postgraduate School
Monterey, California 93940

15. SECURITY CLASS. (of this report)
Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)
Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)
Aircraft Accident Investigation
Flight Data Recording
Magnetic Bubble Memory
Microcomputers

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)
The design, breadboard implementation and functional testing of a Digital Flight Data Recorder/Monitor prototype is reported. Microcomputer interfacing to Magnetic Bubble Memory and a digital data bus was accomplished. The microcomputer is to be used to collect and analyze flight parameters and record only significant data for later use in an accident investigation or maintenance

debriefings for accident prevention. The Magnetic Bubble Memory is the nonvolatile recording media. The digital data bus interface to the IEEE 488 instrument bus will allow operational testing of the recorder on an aircraft equipped with a complete data acquisition system.

Approved for public release; distribution unlimited.

Microcomputer Based Flight Data Recorder/Monitor with Solid State Memory

by

Darl Eugene Easton
Captain, United States Air Force
Bachelor of Aeronautical Engineering
Embry-Riddle Aeronautical Institute, December 1968

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1978

ABSTRACT

The design, breadboard implementation and functional testing of a Digital Flight Data Recorder/Monitor prototype is reported. Microcomputer interfacing to Magnetic Bubble Memory and a digital data bus was accomplished. The microcomputer is to be used to collect and analyze flight parameters and record only significant data for later use in an accident investigation or maintenance debriefings for accident prevention. The Magnetic Bubble Memory is the nonvolatile recording media. The digital data bus interface to the IEEE 488 instrument bus will allow operational testing of the recorder on an aircraft equipped with a complete data acquisition system.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to all who have provided background material and technical guidance in the areas of magnetic bubble memories, microcomputers, and digital data bus interfacing.

Mr. Rodney Wingrove and Mr. Charles Jackson from NASA Ames Research provided useful information in the flight data recording field.

Mr. Charles Stuart from Texas Instruments provided technical expertise in the operation of the Magnetic Bubble Memory Module. Without his help and guidance it is doubtful that the author would have had an operational Magnetic Bubble Memory System.

Prof. Uno Kodres, my thesis advisor, provided invaluable guidance and direction for this research.

Finally, I owe much of the credit to my wife, Jennifer, whose patience and understanding provided the moral support required to carry out this project.

I. INTRODUCTION

A. BACKGROUND

A Flight Data Recorder (FDR) can be an invaluable asset to an accident investigation team. An analysis of recorder parameters can aid the team in determining the exact cause, identifying contributing events, and establishing corrective measures. In a large majority of accidents, this analysis can be done more quickly, more thoroughly, and less expensively if the investigation team has data from a FDR available. Additionally, if the data is in a crash survivable package, it may be available for analysis when ordinarily all other evidence of the accident may have disintegrated in fire or sunk to depths preventing recovery.

Unfortunately, to date, the large majority of operational aircraft do not have an FDR installed. The size, weight, cost, and adaptability of recording systems has restricted their use in all but the large commercial and military transport aircraft.

A recent letter from the National Transportation Safety Board to the Federal Aviation Administration points out the growing need for flight recorder devices on high performance fixed wing aircraft. The following is quoted from that letter [Ref. 24]:

> "The National Transportation Safety Board is concerned about the number of accidents involving complex fixed wing, multi-engine aircraft in air taxi and corporate/executive operation in which the accident

circumstances remain unknown. Of the 194 fatal accidents in these operations from 1970 to 1977, cause has not been determined for 34 of the accidents.

"The value of the FDR, and in particular the digital FDR, has become evident in the investigation of a number of air carrier accidents in which wind shear was a primary causal factor. The recorded data have provided a means for accurately determining the flight profiles and the direction and magnitude of winds. They have also provided sufficient information for programming aircraft simulators so that the condition encountered by the pilots could be reproduced in real time. Simulation based on FDR data has made it possible to explore human factors such as restricted visual cues which hinder prompt recognition of a developing descent rate and accurate assessment of pitch attitude change required to arrest the descent before impact."

The earliest FDRs used electro-mechanically operated styli to scribe analog data on a metal foil recording medium. When digital sensing equipment became available for installation and the digital data became available, digital flight data recorders (DFDR) were built using magnetic tape or steel wire as the bulk storage medium.

The typical foil recording device weighs in excess of 20 pounds and requires approximately 700 cubic inches for installation. Digital flight recorders are nearly the same size and weigh two to four pounds more. The digital recorders generally require a separate data acquisition unit to condition the signals and this unit weighs from 18 to 20 pounds and will require additional space for installation.

Current technology in micro-processor based data processing units and solid-state, nonvolatile mass memories

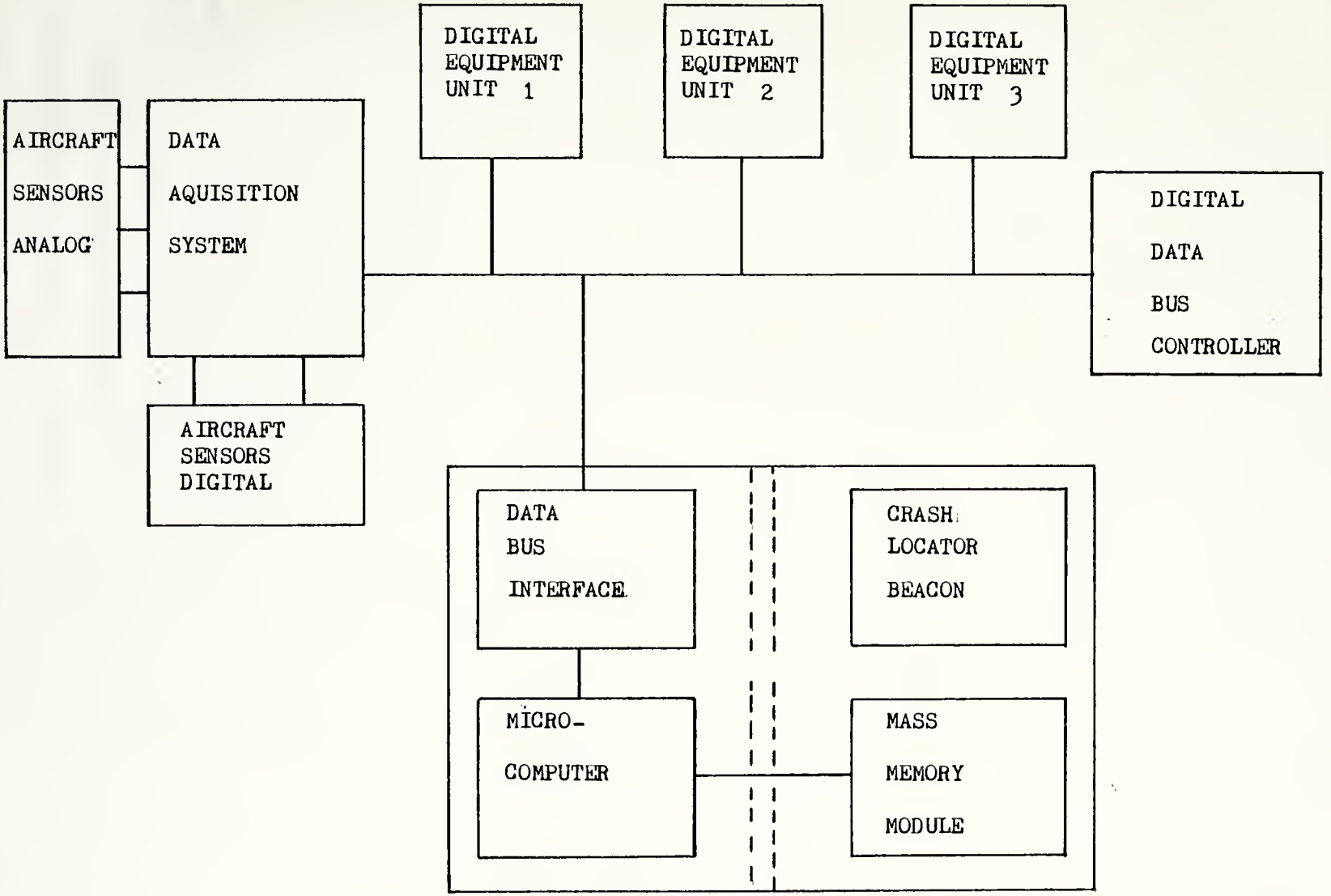
suggests that a crash survivable DFDR can be designed and built with a substantial reduction in cost, size, and weight. The proposed design would incorporate a microcomputer system to collect the digital flight parameters and control the recording of the desired parameter into a solid state memory unit such as a Magnetic Bubble Memory (MBM) unit or Electronically Alterable Read Only Memory (EAROM). In addition, the microcomputer could be programmed to eliminate the storage of redundant data to further reduce the size of the required memory and also provide a means to dynamically analyze parameters to provide automated warnings or commands for the pilot and/or crewmembers.

B. DATA ACQUISITION AND RECORDING

Figure 1 is a functional diagram of a complete digital acquisition and recording system. The particular characteristics of each component would be dependent on the aircraft in which it is utilized.

The data acquisition unit would receive analog and/or digital data from the various aircraft sensors. It would convert the analog signals to digital form and multiplex this information together with directly available digital data onto the universal digital data bus as directed by the bus controller.

The digital data bus controller is used to coordinate the bus usage by the system. It must have the capacity to resolve conflicts and establish priorities for the use of



Digital Acquisition and Recording System
Figure 1

the bus. The digital data bus would interconnect all devices which have the capability and the need to receive information from one or more similarly compatible units. The bus structure and protocol would be highly dependent on the complexity of the communications desired.

The Digital Flight Data Recorder/Monitor (DFDR/M) can be functionally divided into four components:

(1) Microcomputer
(2) Mass memory module
(3) Crash locator beacon
(4) Data bus interface

Only the electronic interface to the digital data bus would have to be tailored specifically for each application, the other components would remain the same. Depending on the application, the DFDR/M may be required to send a request for service to the bus controller or it may receive an interrupt signal to indicate that data is available. Regardless of the algorithm used, the data from the data acquisition must be received by the DFDR/M unit at precisely timed intervals.

After the DFDR/M has received a block of data of the various parameters, it must perform data analysis to determine what data values need to be stored into the nonvolatile storage unit. If the algorithm is not too complex and the number of parameters received is reasonable, considerable computing time may be available for monitoring and correlating the collected parameters. By just monitoring the interrelation of aircraft airspeed, altitude, and attitude,

the DFDR/M could provide automated warnings of impending critical flight conditions.

The data mass-storage unit must be a nonvolatile recording medium. Solid state memory devices under study for this application were MBM modules and several other devices using MNOS transistor technology [Refs. 2 and 27]. The mass memory unit must be located in a crash survivable package.

Collocated in the crash survivable package would be a crash locator beacon. The locator beacon would be activated by crash sensors to transmit an emergency radio signal for an extended length of time to aid search and rescue personnel in locating the accident site.

The technique used to assure crash survivability of the data storage and locator beacon module will be highly dependent on the operational use of the aircraft on which the unit is to be installed. Ideally, it would be an ejectable package which would separate from the airframe when certain acceleration or temperature limits are exceeded. However, such a configuration will add weight and would require extensive modifications to the airframe of most currently operational aircraft. Other methods for impact and fire protection have been investigated and are reported in Ref. 27.

C. OBJECTIVE AND OVERVIEW

This thesis was one part of a continuing research program at the Naval Postgraduate School to design, build, and

test a solid state Digital Flight Data Recorder/Monitor prototype. References 1 and 2 investigated various recording algorithms. Digital flight data tapes were obtained from the National Transportation Safety Board for the testing of the data reduction algorithms. The conclusions of these tests were that a microprocessing unit with as little as 100,000 bits of solid state storage would maintain at least ten minutes of flight data for a 20 parameter recording. Longer periods would be possible under cruising flight conditions since only those parameters which exceed a predetermined tolerance would be recorded.

Reference 3 presented a preliminary design of a DFDR prototype. The prototype utilized commercially available components to reduce development cost. The hardware components were partially constructed with some operational test of the MBM module completed.

The objective of this thesis was to complete the magnetic bubble memory interface circuitry and design and construct interface circuitry to a digital data bus to allow inflight testing of the prototype.

The digital data bus interface circuitry was designed to be connected to the IEEE-488 digital instrument bus. This design decision was based on informal arrangements with the National Aeronautics and Space Administration's Ames Research Center to flight test the DFDR/M prototype on a Cessna 402 aircraft equipped with a data acquisition system utilizing the digital bus for the transmission of

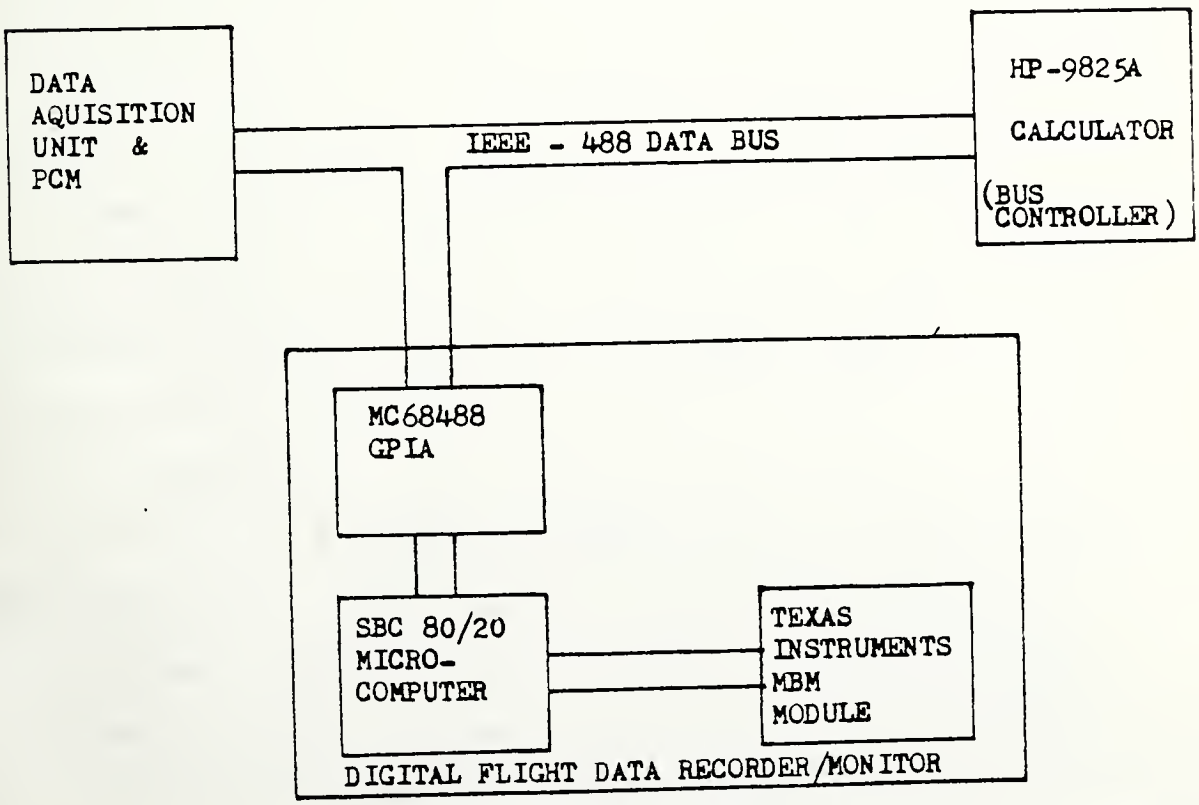
data to a HP-9825A calculator which functions as a bus controller and recorder. This configuration would also be used for testing the recording algorithms prior to flight testing by simply using the HP-9825A calculator to transmit data from previously recorded data tapes in much the same format that the data would be received from the data acquisition unit.

Section II of this thesis reports on the equipment used to construct the prototype and also gives a description of the HP-9825A calculator. Section III provides a description of the equipment used for software development and hardware testing of the prototype. Appendix A is a brief description of the IEEE-488 data bus. The bus structure and message transmission protocol is outlined. Appendix B is a description and listing of computer programs written to test the various programmable hardware interfaces for the MBM module and the data bus. Appendix C lists the minor changes made to the hardware components for utilization in the project. Appendix D, which includes several examples, is a tutorial on the development system operation.

II. PROPOSED FLIGHT DATA RECORDING SYSTEM

A. GENERAL

The Digital Flight Data Recorder/Monitor prototype specified in this thesis was designed to interface to the IEEE 488 instrument data bus to allow functional testing of the prototype on a flight test aircraft. Figure 2 shows the functional breakdown of the integrated system. The aircraft, a Cessna 402 operated by the NASA Ames Research Center, was equipped with a data acquisition system and a programmable Pulse Code Modulator (PCM). The HP-9825A calculator functions as the IEEE 488 instrument bus controller and can be used to program the PCM. Programming data to the PCM includes time framing information and data channel selection.



DFDR/M PROTOTYPE SYSTEM
Figure 2

The PCM, once programmed, would collect the digital data requested from the various channels, issue a service request to the bus controller, and subsequently send the data as a bus talker. The HP-9825A calculator functioned as the bus controller and was responsible for addressing the DFDR/M as a bus listener prior to addressing the PCM. See Appendix A for the IEEE 488 bus protocol.

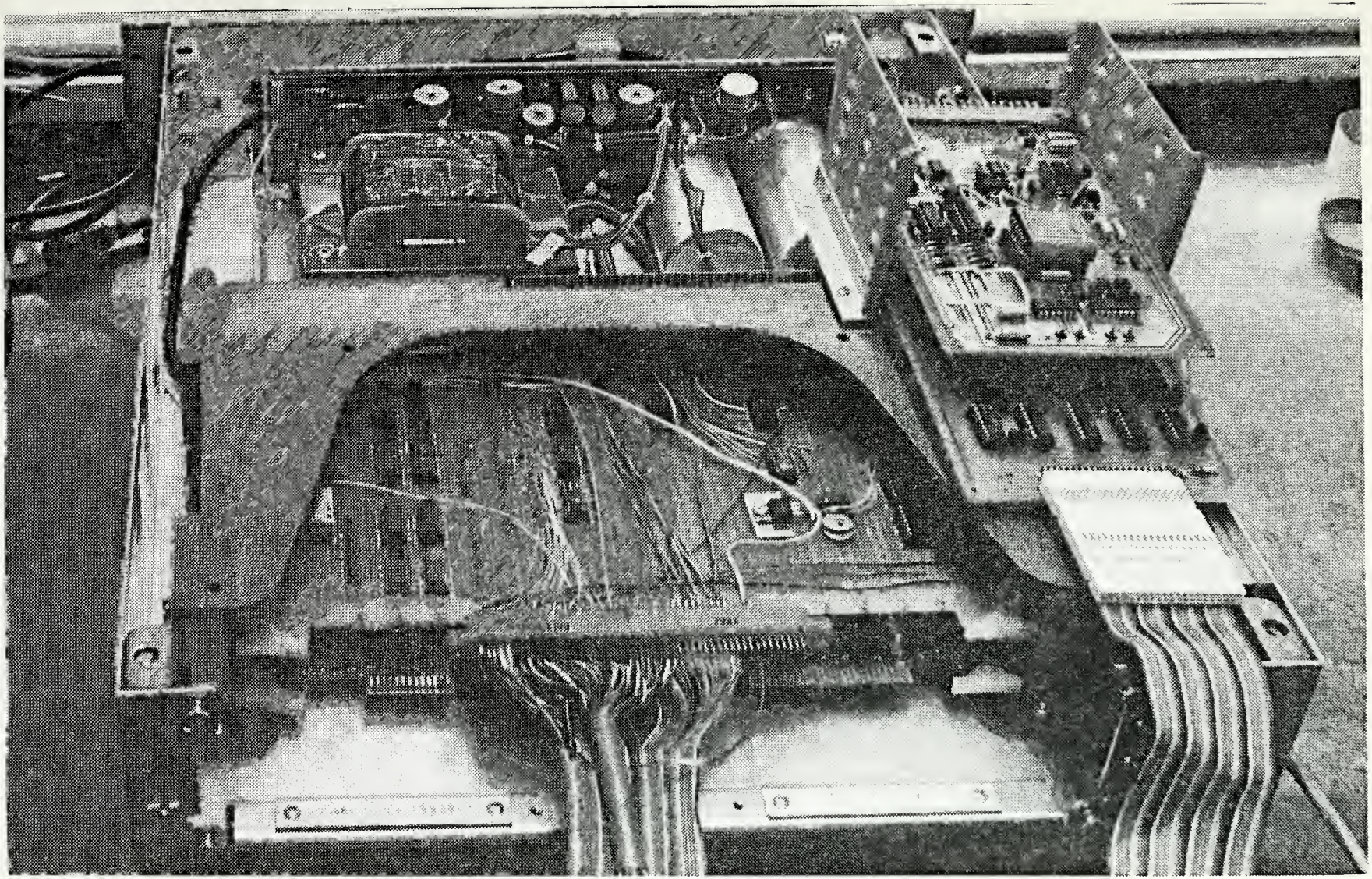
This section will describe the components of the DFDR/M system and the data bus controller. The operational information of the data acquisition system and PCM is not described in detail.

B. DIGITAL FLIGHT DATA RECORDER/MONITOR

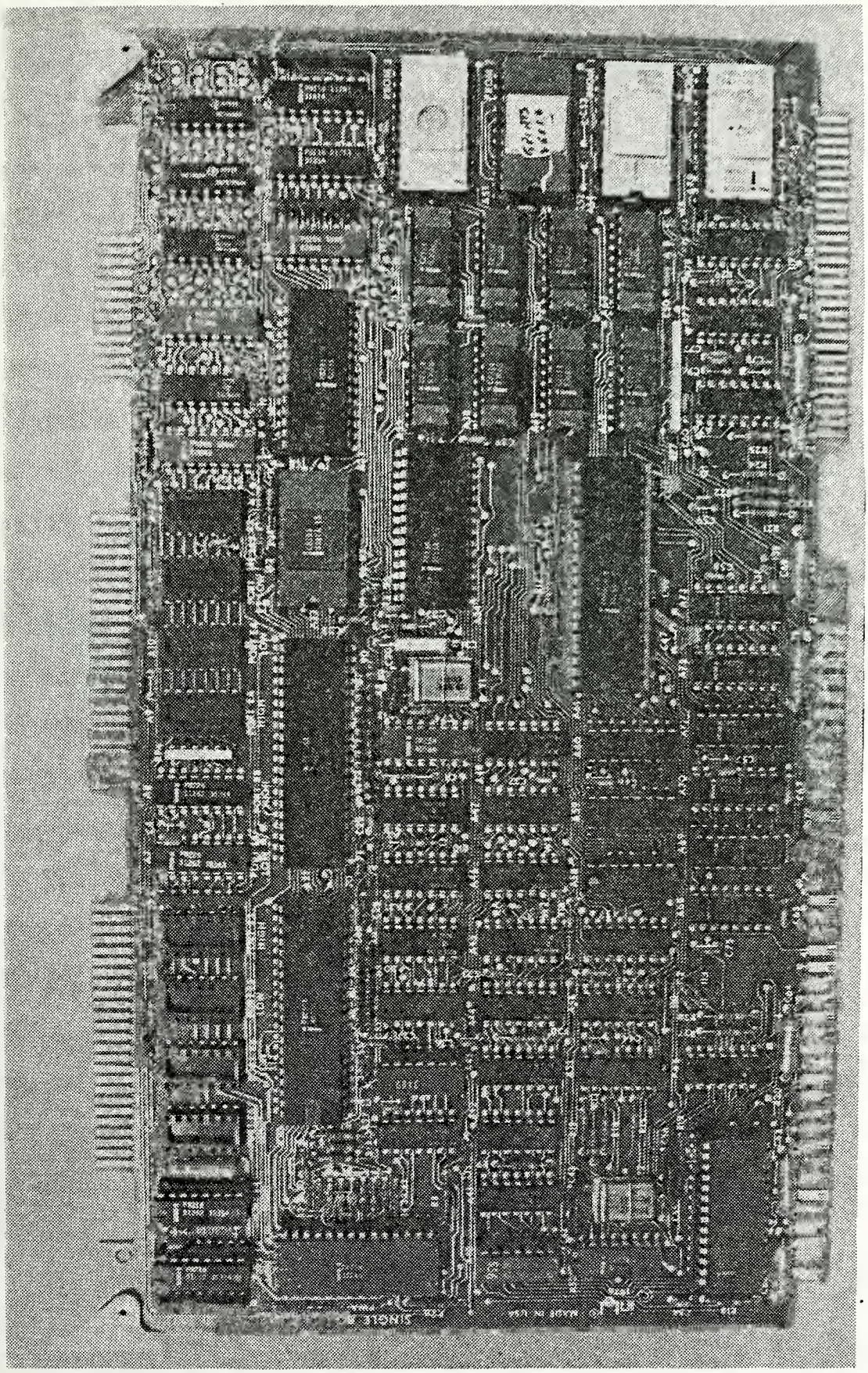
The DFDR/M prototype is shown in Figure 3. The system components are categorized in Figure 2. Data transmitted to the DFDR/M would be received via the MC68488 General Purpose Interface Adapter (GPIA), processed by the SBC 80/20 microcomputer, and pertinent flight data recorded into the Texas Instruments Magnetic Bubble Memory module. The description of the hardware components of the DFDR/M will detail their functional purpose as well as the element design characteristics.

1. SBC 80/20

Figure 4 pictures the SBC 80/20. The SBC 80/20 was the programmable microcomputer in the DFDR/M prototype. The unit was used to program the MC68488 GPIA and the TMS 9916 MBM controller. However, its primary role was to process the data parameters as received, compare the most



DIGITAL FLIGHT DATA RECORDER/MONITOR PROTOTYPE
Figure 3



SBC 80/20 SINGLE BOARD COMPUTER
Figure 4

recent value with previously stored values and/or keep a running total of deviations from the last recorded value to determine if a particular parameter needed to be stored in mass memory. This real-time processing of parameter data allows more information to be compressed into the nonvolatile mass memory, resulting in more effective utilization of the limited memory space.

The SBC 80/20 was Intel Corporation's commercially available single board computer based on the Intel 8080A-2 CPU and its associated family of support chips. The system has 4K bytes of Random Access Memory (RAM) and 4K bytes of Eraseable Programmable Read Only Memory (EPROM). 2K bytes of EPROM was used for the SBC 80/20's monitor program. The monitor provided the necessary software to communicate with any RS232 compatible communication terminal. It also provided the capability to enter and modify machine code (hexadecimal) programs.

The SBC 80/20 provided a wide selection of hardware support for input/output (I/O) and interrupt structuring. It had two 8255 Programmable Peripheral Interface (PPI) chips and one 8251 Programmable Communication Interface (PCI). The PPI could be used in a wide variety of ways for programmable control of I/O ports. For a detailed description, see Ref. 19. The PCI is primarily used for bit-serial communication with terminals and line printers.

Interrupt structuring was provided for by an 8259 Programmable Interrupt Controller (PIC). The PIC allowed

considerable flexibility in the interrupt structure and was software controllable to allow dynamic structuring [Ref. 19]. A memory map table and a list of minor modifications made to the SBC 80/20 are found in Appendix C.

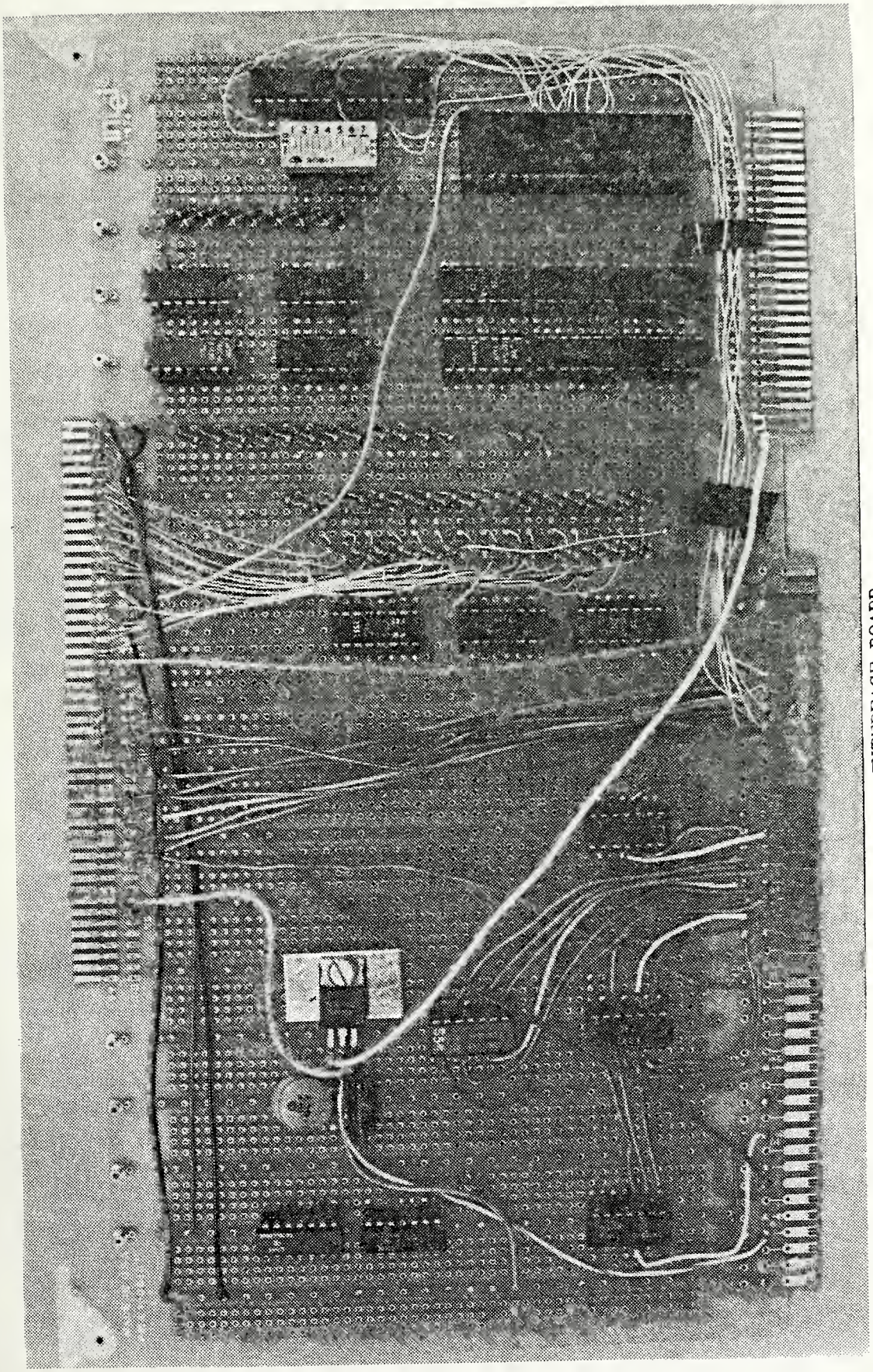
2. Digital Data Bus Interface

The DFDR/M prototype system was designed to interface to the IEEE 488 instrument data bus. The bus interface components were assembled on the general purpose design board shown in Figure 5. A functional diagram of the bus interface is shown in Figure 6 and the detailed wiring diagrams are in Appendix C.

The heart of the bus interface was the LSI/MNOS Integrated Circuit chip manufactured by Motorola. The MC68488 GPIA was designed to provide a means to interface between the the IEEE 488 instrument bus and the MC6800 CPU. However, due to similarities in the data and address bus structures of the 8080 CPU and the MC6800 CPU, compatibility between the SBC 80/20 (an 8080 CPU based single board computer) and the MC68488 GPIA was obtained with some use of external logic.

Reference 9 was an advanced information data sheet for the MC68488 GPIA. Reference 10 gave a simplified description of the IEEE 488 instrument bus and provided a sample assembly language program demonstrating the use of the MC68488.

The MC68488 GPIA provided the majority of the bus interface functions specified by the IEEE 488 instrument

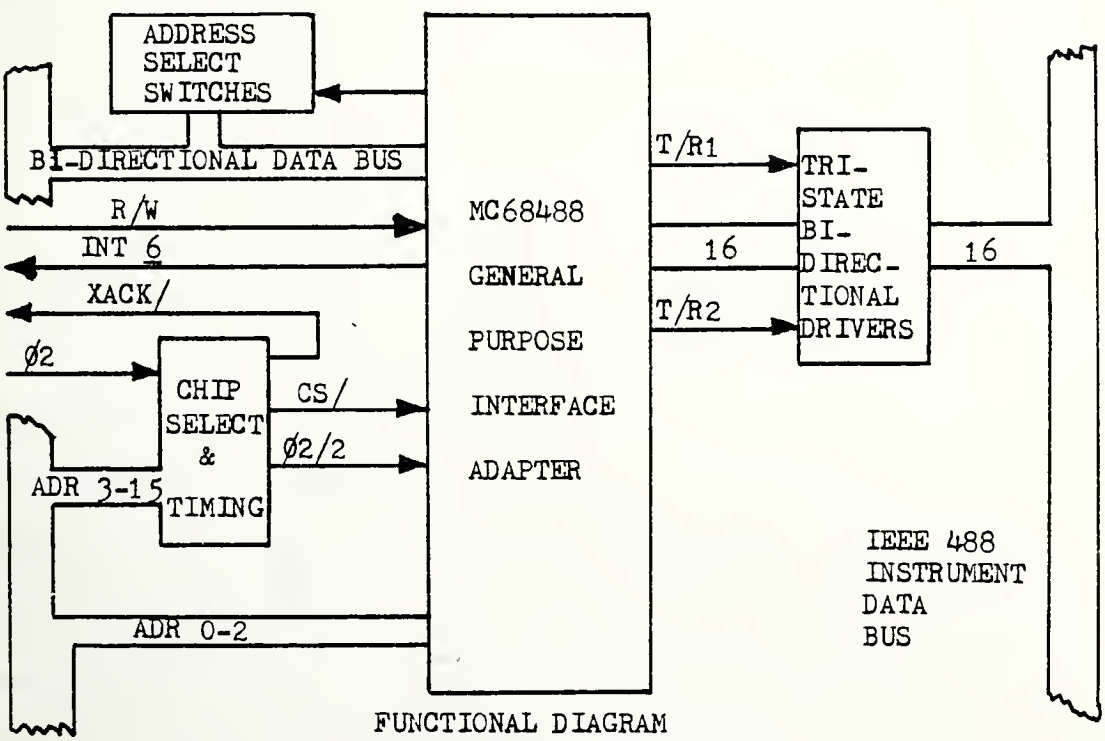


INTERFACE BOARD
Figure 5

bus standard [Ref. 8] and by selectively programming it, the system designer can control and monitor the operation of the device. The major features of the MC68488 GPIA are:

(1) Single and dual primary addressing
(2) Extended addressing
(3) Complete Source-Acceptor handshake
(4) Programmable Interrupts
(5) Ready for Data holdoff
(6) Serial and parallel polling
(7) Listen and Talk interface functions

The MC68488 GPIA's chip select and 02 clock signals were provided by the address decode and timing circuitry. The high order address lines from the SBC 80/20 were used to memory map the MC68488 GPIA to memory addresses FFE0H through FFE7H. The chip select signal generated was synchronously applied to the MC68488 GPIA and the Transfer Acknowledge



FUNCTIONAL DIAGRAM
DIGITAL DATA BUS INTERFACE
FIGURE 6

(XACK/) signal was returned to the SBC 80/20 when data set-up and hold times of the two devices were compatible.

The three low order address lines were connected directly to the Register Select pins (RS0-RS2) of the MC68488 GPIA. The GPIA's internal registers provided for programmable control and status monitoring of the device. The tri-state, bi-directional data bus of the GPIA was connected directly to the data bus of the SBC 80/20 to transfer data between the internal command and status registers of the MC68488 GPIA and the working registers of the 8080 CPU.

The 02 clock to the GPIA was one-half the frequency of the 02 clock of the 8080 CPU on the SBC 80/20 board. The read/write (R/W) control signal was connected directly to the SBC 80/20/s Memory Write Control (MRWC/) signal.

The GPIA was connected to the IEEE 488 instrument data bus through the signal line buffering circuitry. This circuitry provided bi-directional tri-state drive capability to twelve of the interface lines. All of the signal lines required resistive termination. Directional control of the drivers was provided by the Transmit/Receive (T/R) signals from the GPIA.

The DFDR/M's address for IEEE 488 bus operation was determined on initialization by reading the address select switches. The five low order bit positions of the switches determined the talk or listen address of the device. The high three order bits can be user defined and one

application would be to use them to select between a talk only or listen only mode of operation for the GPIA.

3. Magnetic Bubble Memory Unit

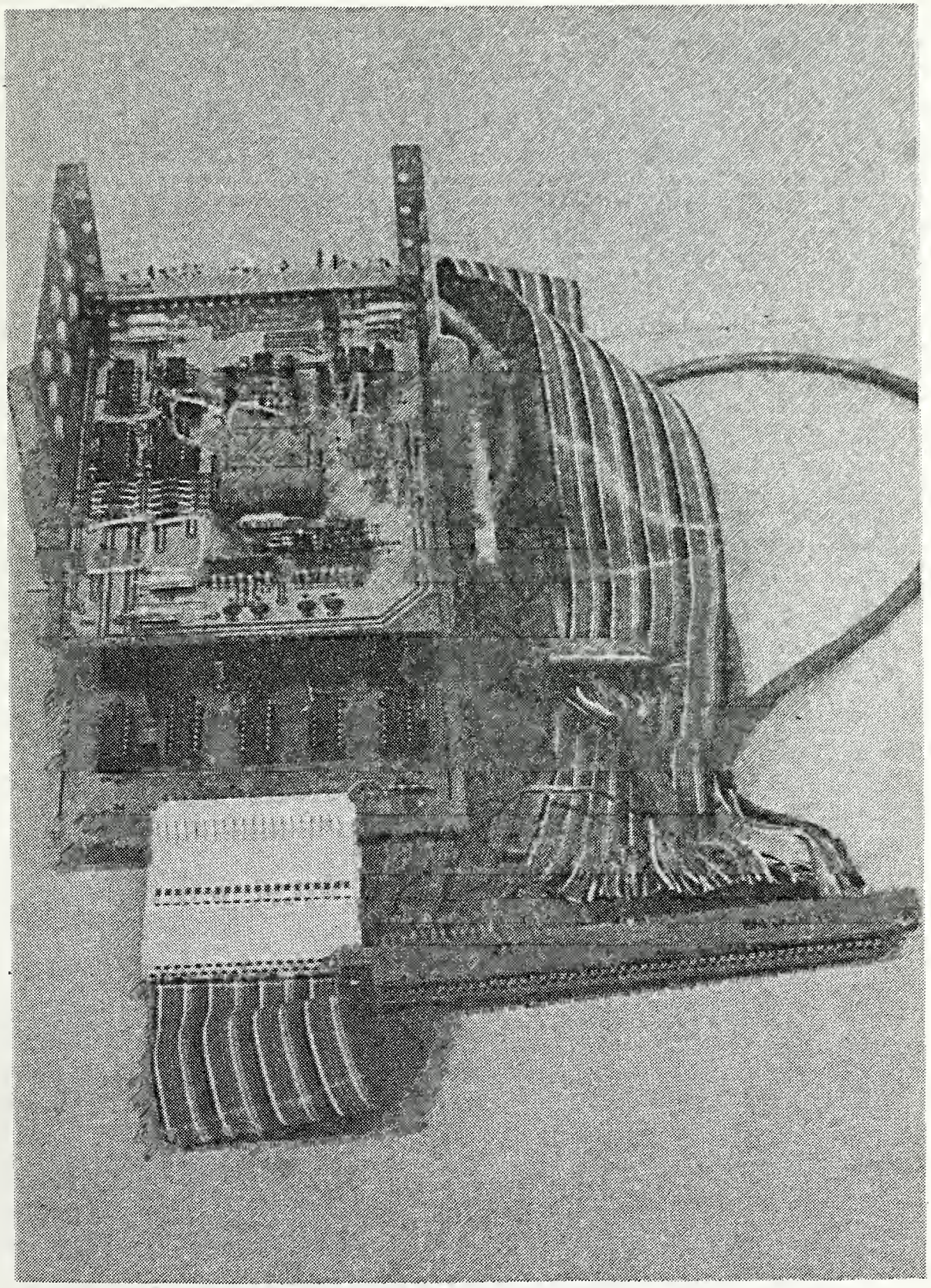
The Magnetic Bubble Memory (MBM) unit is pictured in Figure 7. It consists of two printed circuit boards (MBM controller board and MBM board), the card cage, and the connecting cabling. References 6 and 7 provided the logic and schematic diagrams of the MBM controller board and the MBM board respectively. A functional diagram for each is included in this section. The cable leads are tabulated in Table V of Appendix C.

a. MBM Controller Board

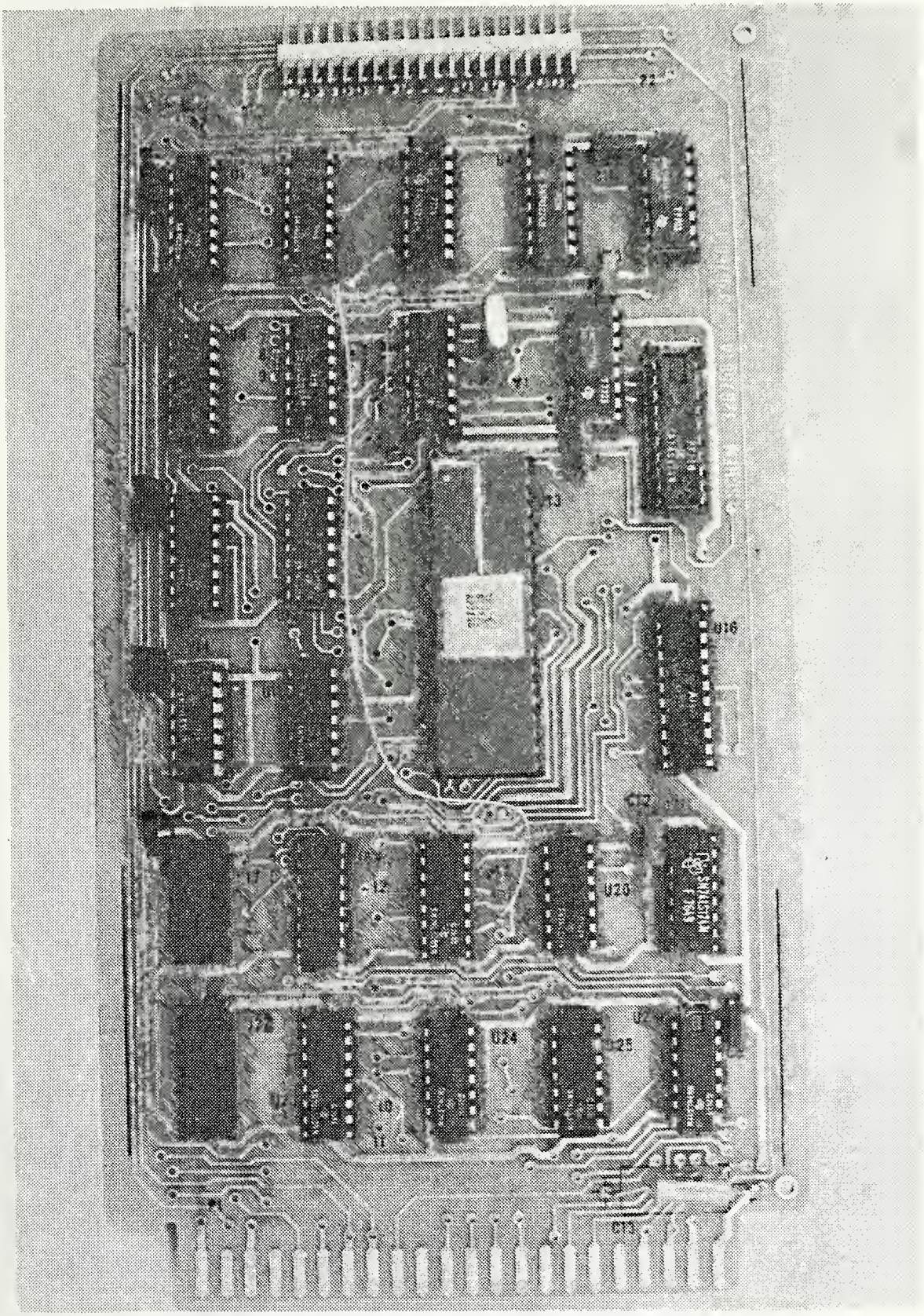
The MBM controller board is pictured in Figure 8 and a functional diagram is detailed in Figure 9. The MBM controller board contains the circuitry required to memory map the MBM controller into the SBC 80/20's addressable memory range. The controller's command and control registers are mapped into memory locations 0FFF0H through 0FFFEH.

The chip select and read/write function decoded the high order address lines and the interface board's MEMEN/ and DBIN signals to generate the chip select (CS/), read (READ/), and write (WRITE) signals to the MBM controller. A synchronization signal for the controller was provided by the clock generation and synchronization function.

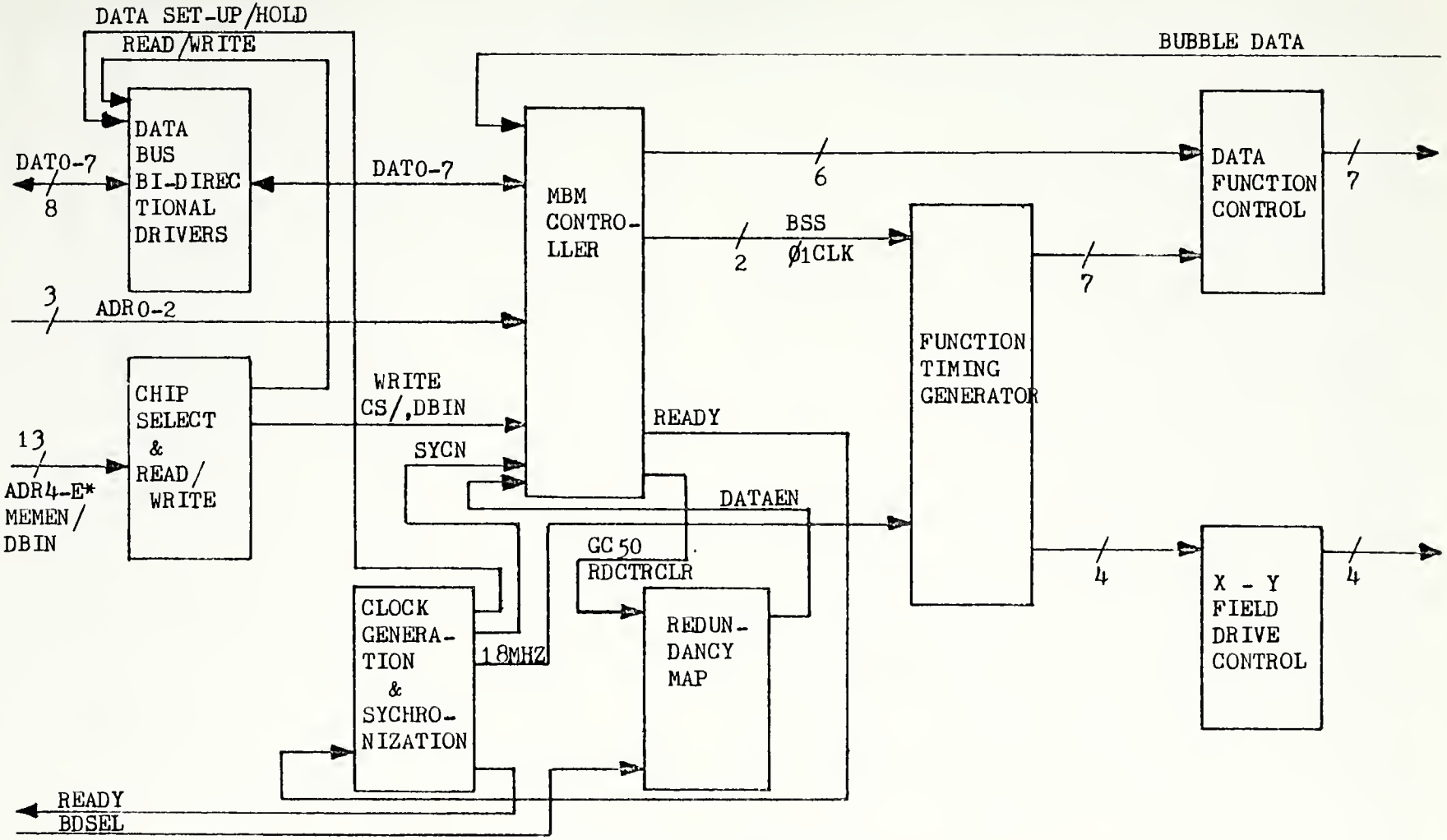
Data bus compatibility was provided by the tri-state bi-directional bus drivers. The drivers receive



MAGNETIC BUBBLE MEMORY UNIT
Figure 7



MAGNETIC BUBBLE MEMORY CONTROLLER BOARD
Figure 8



FUNCTIONAL DIAGRAM
MAGNETIC BUBBLE MEMORY CONTROLLER BOARD
FIGURE 9

direction of data transfer information from the Read/Write circuitry and data set-up and hold timing from the controllers ready signal.

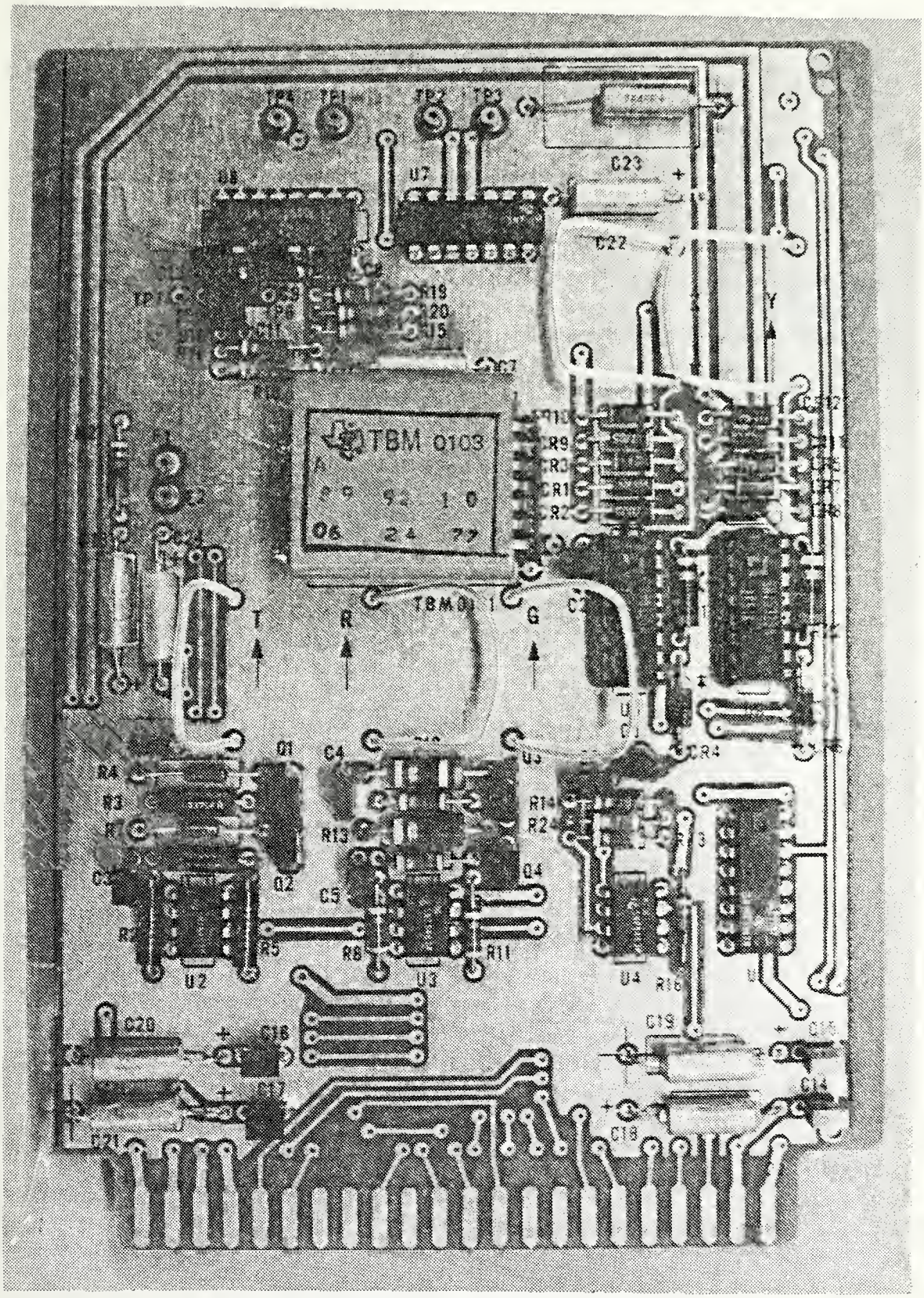
The MBM controller transferred data to and from the MBM by sending control signals to the Function Timing Generator (FTG), the Redundancy Map and the Field Drive and Data Function Control blocks. The FTG guaranteed the proper timing of the coil drive signal and the generate, replicate, annihilate, transfer in, and transfer out function signals.

The clamp and strobe signals are also sequenced through the FTG. These signals were used to sense the presence of a magnetic bubble in a particular location of the MBM.

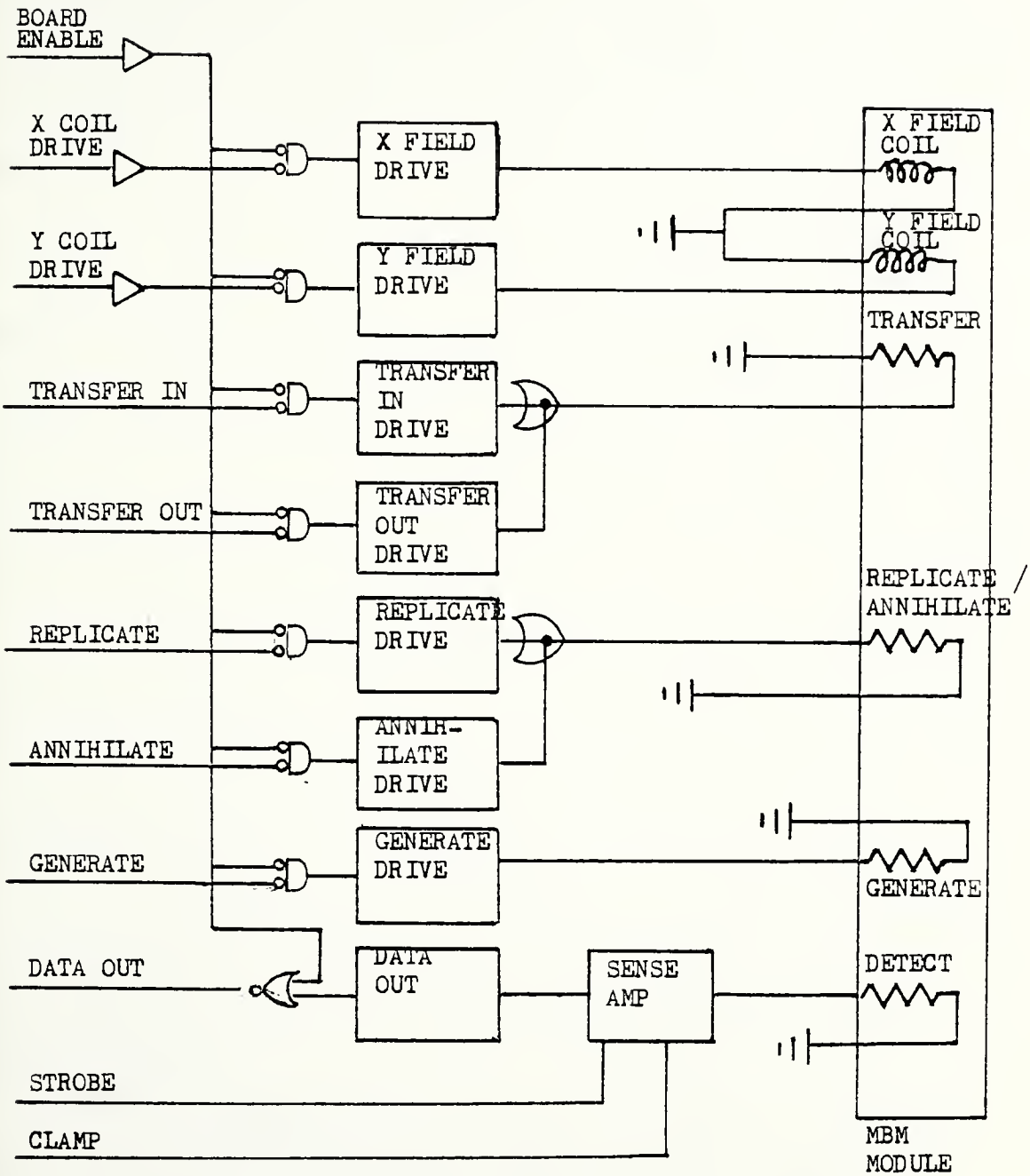
All of the MBM control signals are routed off the MBM controller board to the MBM board. Data returned from the MBM during a read operation was transmitted directly to the controller.

b. MBM Board

The MBM board is pictured in Figure 10 and a functional diagram is detailed in Figure 11. The MBM board contains the circuitry to convert the TTL signals from the MBM controller board into the current level signals required to effect the various functions. A detailed description of the MBM board's operation was found in Ref. 4. An overview of the functional components of MBM board is provided here.



MAGNETIC BUBBLE MEMORY MODULE BOARD
Figure 10



FUNCTIONAL DIAGRAM
MAGNETIC BUBBLE MEMORY BOARD
FIGURE 11

The Board Enable signal was used to collectively control the coil field and bubble function signal drivers. Since only one board was used in this application, this signal was input low to permanently enable the unit. If multiple MBM boards were to be used, external logic from the microcomputer unit would be required to enable a particular memory module.

Field drive control signals were received from the MBM controller board. The precisely timed input signals were applied to appropriate coil drivers to form a sawtooth current waveform. The combined effect of the X and Y field drives was a rotating magnetic field. The rotation of this magnetic field was used to control the positions of the magnetic domain bubbles [Ref. 4].

The transfer in and transfer out drivers were functionally identical and drive the same gate in the MBM. The precise timing of the control signals from the MBM controller board determines whether the gating action will cause a transfer in or a transfer out operation to occur.

Similarly, the replicate and annihilate drives are functionally identical and drive a common gate in the MBM. Timing of the signal application determines which action occurs.

The generate circuitry in the MBM is individually controlled by the generate gate driver. The proper sequencing of the control signal came from the MBM controller board.

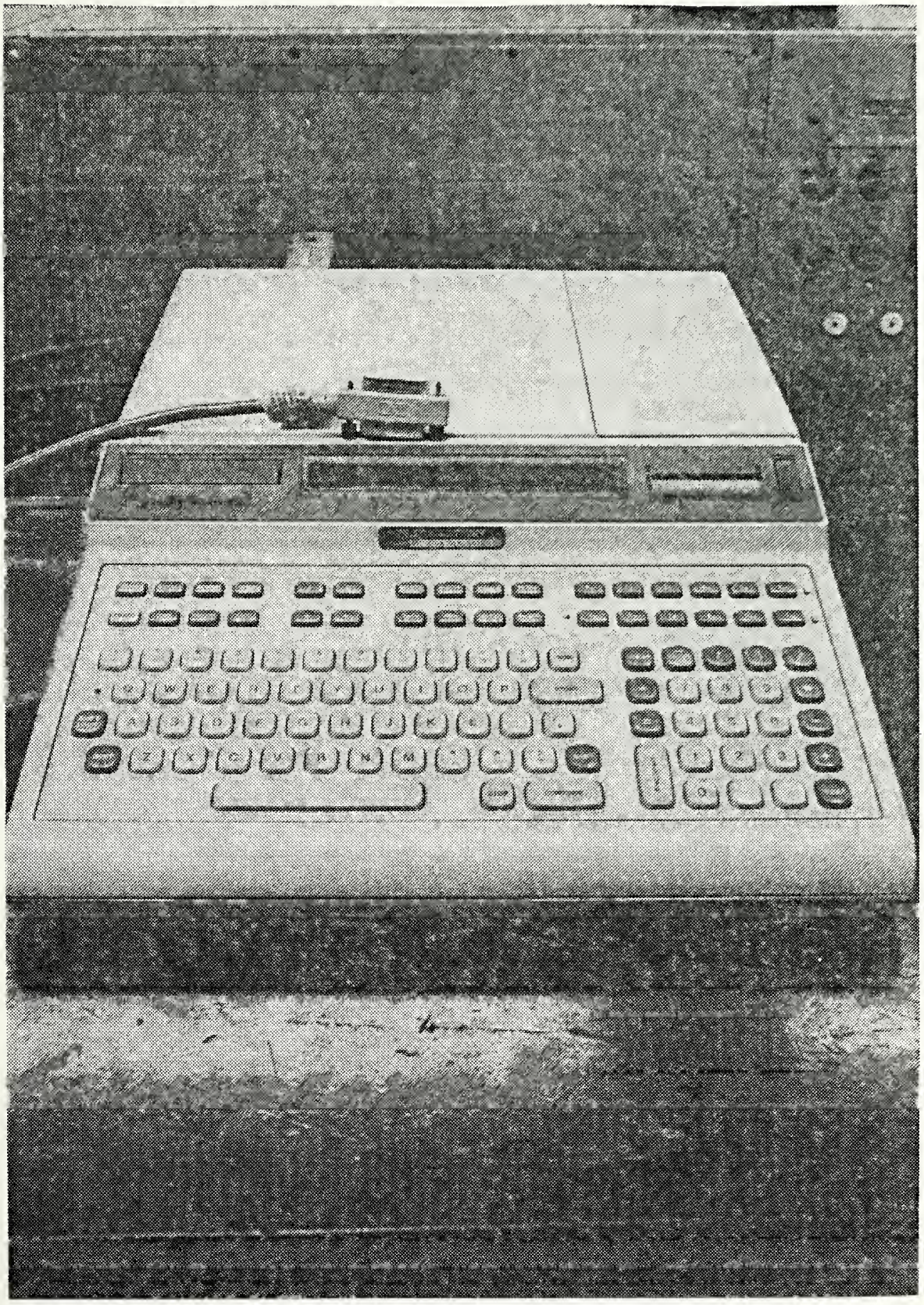
During a MBM write operation the control signal to the generate gate was used to control the presence or absence of a magnetic bubble domain to represent the input data stream. The signal was not used during an MBM read operation.

Data stored in the MBM was recovered using a sense amplifier to receive a millivolt level analog voltage from the magnetic bubble detect elements. The clamp signal was an input to the sense amplifier network to synchronize its action with the transfer of data across the detect element. The strobe signal synchronized the transfer of data information to the MBM controller.

C. DATA BUS CONTROLLER

The Hewlett - Packard 9825A calculator pictured in Figure 12 was used as the digital data bus controller. The calculator provided the necessary bus controls to thoroughly test the interface to the DFDR/M prototype. References 25 and 26 provided operating and programming information. Examples were provided in the manuals on the applications and limitations of the calculator as a bus controller.

The calculator that was available for testing in the laboratory was equipped with 8192 bytes of RAM, a General Extended I/O ROM, and the 9803A interface kit for communication interfacing to the HP-IB interface bus (the HP-IB interface bus is identical to the bus specified in the IEEE 488 standard).



HEWLETT-PACKARD 9825A CALCULATOR
Figure 12

The calculator had a built-in data cartridge recorder. The two track high density data cartridge can be used to store programs and data. Tape cartridges were obtained for the recording of flight parameters during flight testing of the DFDR/M. This will allow for a convenient means to verify the data collected.

Additionally, the HP-9825A can be conveniently connected to a HP-9862A X-Y Plotter. With appropriate formatting of the data files on the cassette cartridge, the flight data collected could be plotted for visual presentation. This feature could also aid in the analysis of the data stored in the DFDR/M.

The MC68488 GPIA interface can also function as a bus talker. Post flight analysis of the recorded compressed data in the MBM could be sent through the HP-IB interface to the calculator for subsequent plotting for visual comparison with uncompressed data stored on the cassette.

III. SOFTWARE DEVELOPMENT AND HARDWARE TESTING

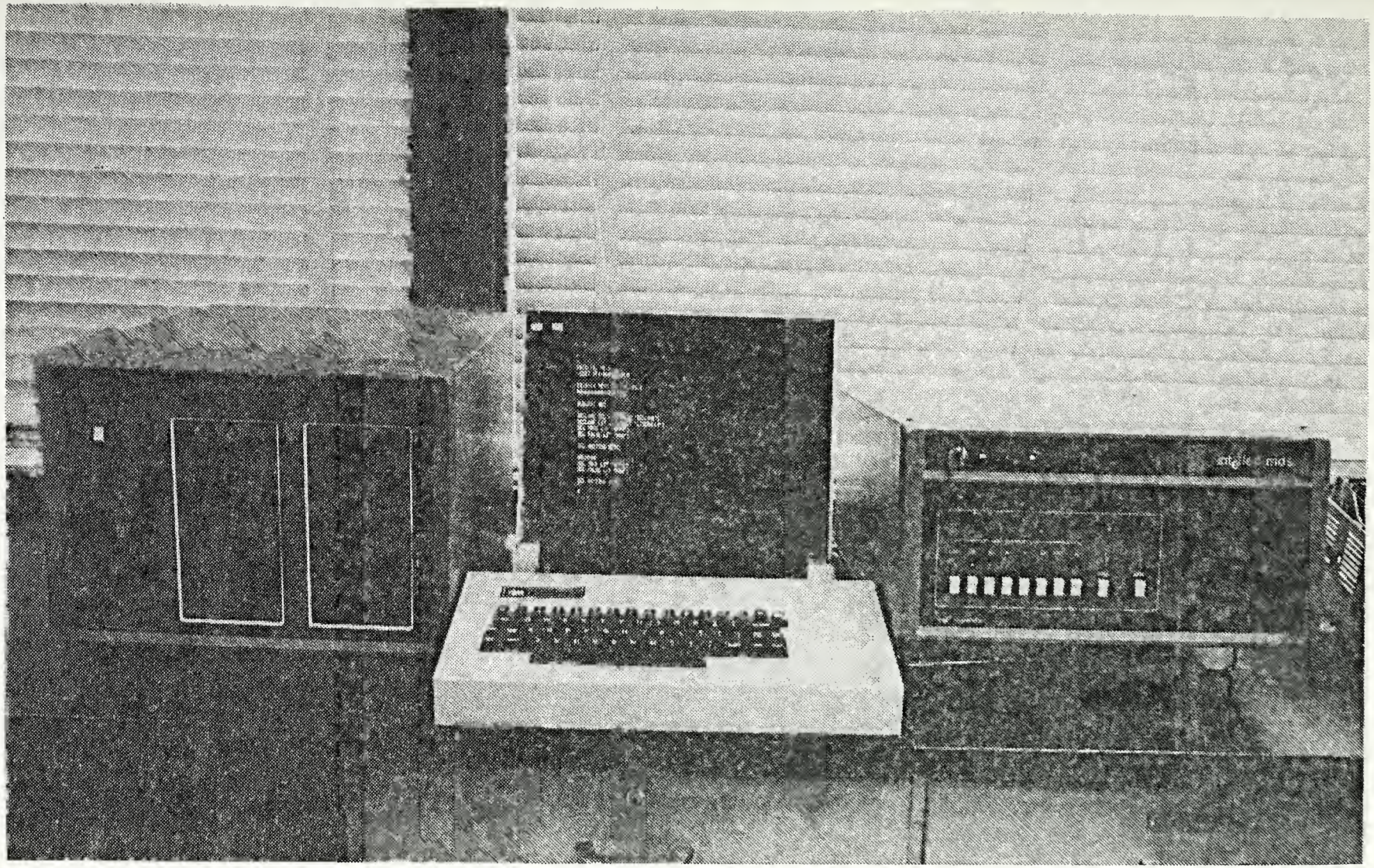
A. DEVELOPMENTAL SYSTEM DESCRIPTION

The system used for software development and hardware testing is shown in Figure 13. The Intellec Microcomputer Development System (MDS) was equipped with a diskette operating system, the In Circuit Emulator 80 (ICE-80) hardware and software, and the Intel System Implementation Supervisor (ISIS-II) software package. The system had 64k bytes of random access memory, a system monitor module, and a front panel control module installed.

B. SOFTWARE DEVELOPMENT

The MDS diskette hardware and the ISIS-II software package combined to form an effective software development tool. Program files were created and edited by the use of a conventional CRT terminal. Programs written in either one of the two systems programming languages, PL/M-80 or ASM-80, are translated into relocatable object code files by the ISIS-II PL/M-80 compiler or the ISIS-II 8080/8085 Macro Assembler (ASM80) respectively. These files can then be linked and processed into executable absolute object code files by the ISIS-II LINK and LOCATE commands.

Appendix B contains PL/M-80 programs developed on the system. Both the GPIA.PLM and the MBM.PLM modules were written, debugged, and then placed in a Digital Flight Data Recorder/Monitor (DFDR/M) system library (AIRDAT.LIB) using



MICROCOMPUTER DEVELOPMENT SYSTEM
Figure 13

the ISIS-II library manager. Each of the modules used the Public procedure declaration to allow the procedures contained therein to be used in other modules or main programs.

The RDWR.PLM module declared the procedures used from the library as External. After compilation of the main program module, the LINK command was used to form one relocatable object module. The LOCATE command was then used with the appropriate parameters to form the desired executable object code.

The HEXMEM.PLM program found in Appendix B is a program example which used the ISIS-II system calls for simplified file management and program exit. The program was written to allow EPROM programming using MDS memory space. Through the use of system calls the program opens a diskette file, reads from the file, and subsequently closes the file after the processing is completed. Console I/O was also done through ISIS-II system calls.

References 11 and 12 provided a complete description of the MDS system and ISIS-II operating system. Appendix D is provided to aid the prospective user in understanding the use of the systems development tools.

C. HARDWARE/SOFTWARE TESTING

The MDS compatible ICE-80 hardware pictured in Figure 14 was used extensively for software debugging and testing. The Tektronix 464 oscilloscope pictured in Figure 15 was used during ICE-80 controlled program execution to debug

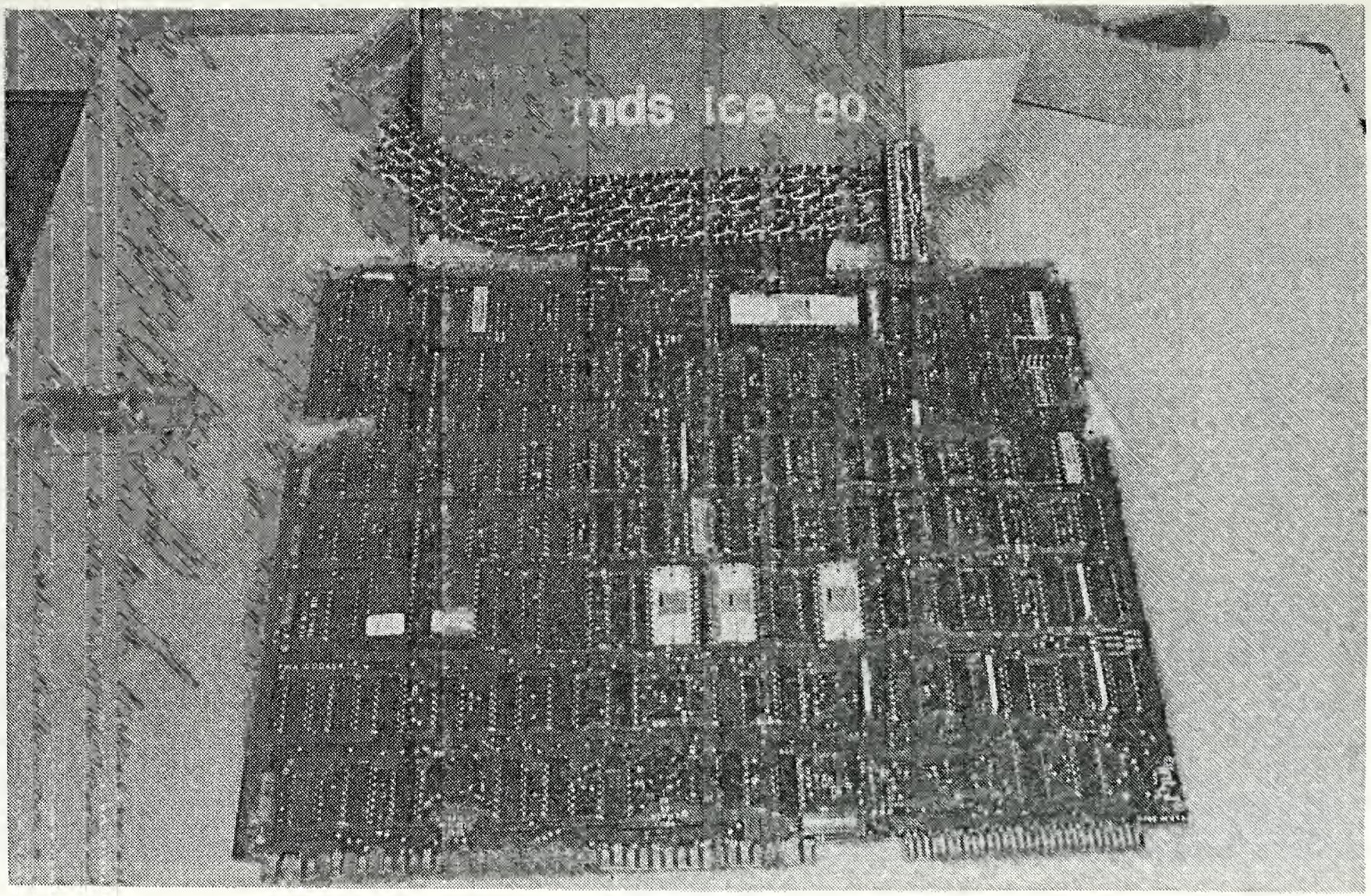
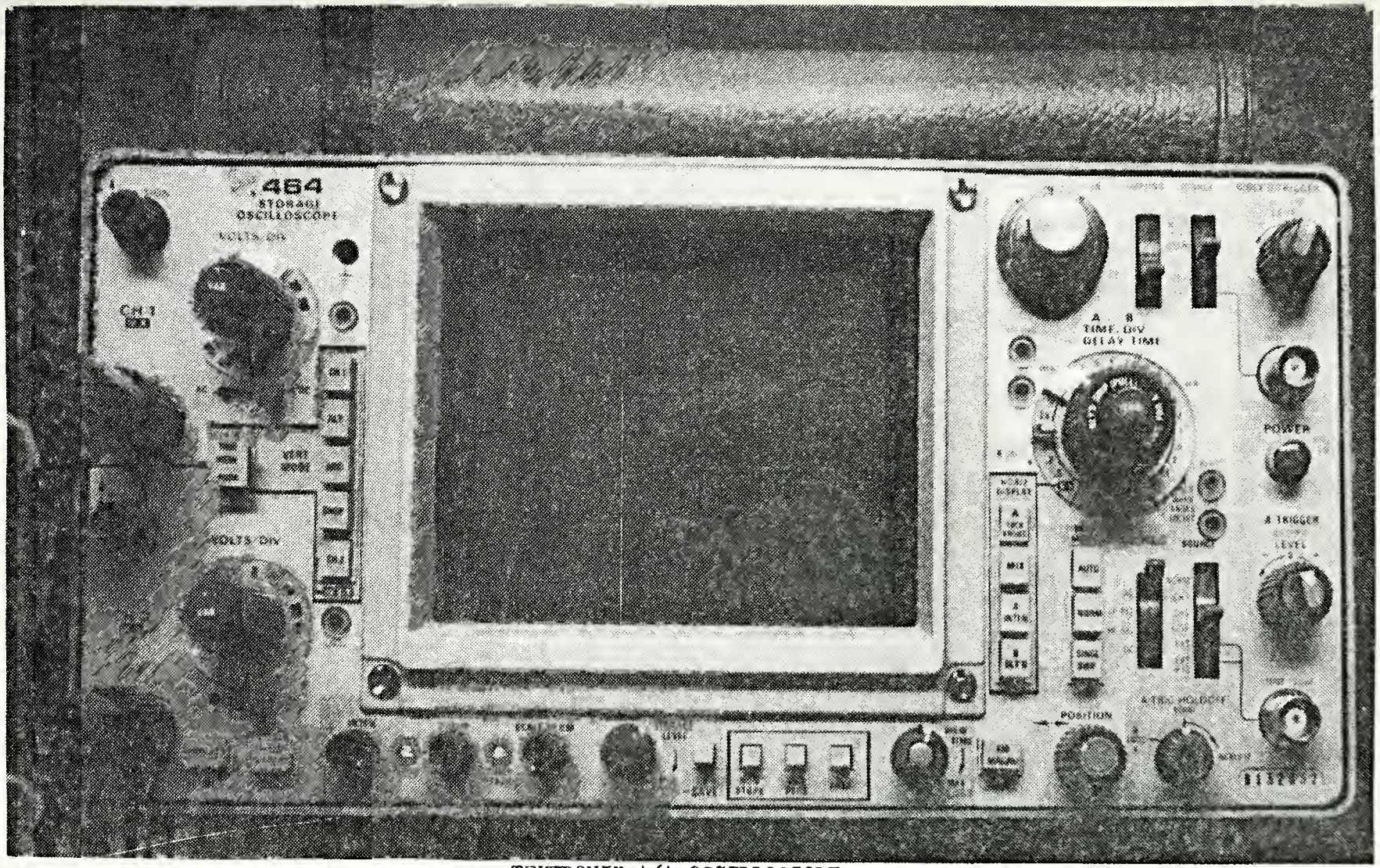


Figure 14



TEKTRONIX 464 OSCILLOSCOPE
Figure 15

and verify the various hardware elements for proper signal timing and voltage levels. Other miscellaneous test and design equipment used were a digital multimeter, a Digi-Designer, an adjustable power supply, and a current probe.

The ICE-80 hardware/software package allowed the emulation of the 8080 CPU and provided program execution control. Because of the nature of the ICE-80, MDS memory and I/O ports could be substituted or used for SBC 80/20 memory and I/O ports. ICE-80 provided the capability to load and execute programs developed on the MDS for the DFDR/M prototype system.

The RDWR.PLM program in Appendix B was written to test the interaction of the SBC 80/20 with the MBM module. After the program was translated, linked and located, the executable absolute object file (RDWR) was loaded through the ICE-80 package into the SBC 80/20's RAM. The program was then tested and evaluated for correctness using the ICE-80 emulation commands. If program errors were detected, the suspect code was displayed and modified using the DISPLAY and CHANGE commands, respectively.

If testing required that the equipment be shut down for modification to hardware elements, the memory contents of a specified range was saved using the ICE-80 SAVE command. This allowed hardware and software testing to resume from a particular point after the modification was completed.

The HEXMEM.PLM program of Appendix B was written to be used in conjunction with ICE-80 to simplify PROM programming

of finalized software. As mentioned earlier, the program uses the ISIS-II operating system calls to load a hexadecimal formatted [Ref. 12] diskette file into MDS memory. ICE-80 was used to map the MDS memory into the SBC 80/20's memory. The PROM programmer [Ref. 22] was used in the SBC 80/20 under control of ICE-80 emulation, to program the EPROM with the machine code previously loaded into MDS memory.

References 15 and 16 provided the details for the installation and operation of the ICE-80 system. Appendix D provides several step-by-step samples of ICE-80 utilization in the design and testing of the DFDR/M.

The oscilloscope was an invaluable testing tool. Numerous hardware construction errors were resolved by tracing the high frequency TTL level signals from the intended source to an erroneous termination. The oscilloscope operation was described in Reference 23.

IV. RESULTS AND RECOMMENDATIONS

All of the hardware components for the Digital Data Recorder/Monitor (DFDR/M) prototype have been constructed and tested. The digital data bus circuitry has been tested as a listener only in both the 'poll status' and 'interrupt' modes and found to be completely operational. Testing should be done to verify the operation of the interface as a bus talker. As a bus talker, the DFDR/M could be programmed to pass data collected during a flight test to the HP-9825A calculator with a X-Y plotter attached to allow post flight analysis.

The Magnetic Bubble Memory (MBM) interface circuitry has also been thoroughly tested. Data, commands, and status information can be sent to and received from the MBM controller with expected results. However, there was a malfunction within the MBM module which was not resolved. Programs which write data to a particular page and immediately read it back out of the MBM have been successfully executed without any errors occurring. Unfortunately, programs which write a unique bit pattern throughout the MBM and then read it back have resulted in a significant number of errors. This problem may be due to an incorrect redundancy map or incorrect sequence and timing of commands to the MBM controller.

The error pattern is somewhat consistent which leads the author to believe that the redundancy map could be

incorrect. The reduncancy map should be reprogrammed to mask out two additional loops. The two loops are adjacent to each other and their addresses can be determined by an analysis of the error pattern returned from the MBM controller.

In addition to correcting the above problem, storage tests need to be conducted to determine if there is any additional loss of data when the MBM remains static for an extended length of time.

A flight data tape cartridge for the HP-9825A calculator was obtained from National Aeronautics and Space Administration's Ames Research Center for preliminary testing of the DFDR/M prototype. This tape included a program to plot the parameters collected on a Hewlett-Packard X-Y Plotter. However, due to the size of the data files, a calculator with a full complement of RAM must be used. The calculator used to test the digital data bus circuitry had less than one-half the memory required to load the files. This problem may be resolved by reducing the size of the data files on a calculator with a larger memory. This action may require obtaining permission from personnel at NASA's Ames Research Center to use the calculator from which the tape was recorded.

Once the data tape cartridge files are reduced in size, the HP-9825A calculator available from the Electrical Engineering Department could be used to simulate the data acquisition system on the flight test aircraft. This will allow complete functional testing of the DFDR/M prototype before attempting a flight test.

Before flight testing of the prototype can begin, formal arrangements with the Ames Research Center must be made. Key personnel there have expressed an interest in the program and have indicated that there should be little trouble in getting the authorization for the experiments. The exact details on the format of the request can be obtained from the Director, Cessna 402 Flight Test Programs, NASA Ames Research Center, Moffett Field, California.

In addition to continued research into various data storage algorithms to obtain an efficient and effective use of the nonvolatile memory storage module, research should be started to design and build a multi-channel data acquisition unit. The availability of low cost single chip eight bit analog to digital converters and the increasing use of digital equipment in aircraft flight directors and navigational systems indicates that a data acquisition unit capable of gathering a significant number of parameters could be assembled into a low cost, light weight package. In all probability, it could be built into the DFDR/M unit with negligible effect on its size and weight.

APPENDIX A

IEEE 488 INSTRUMENT INTERFACE BUS

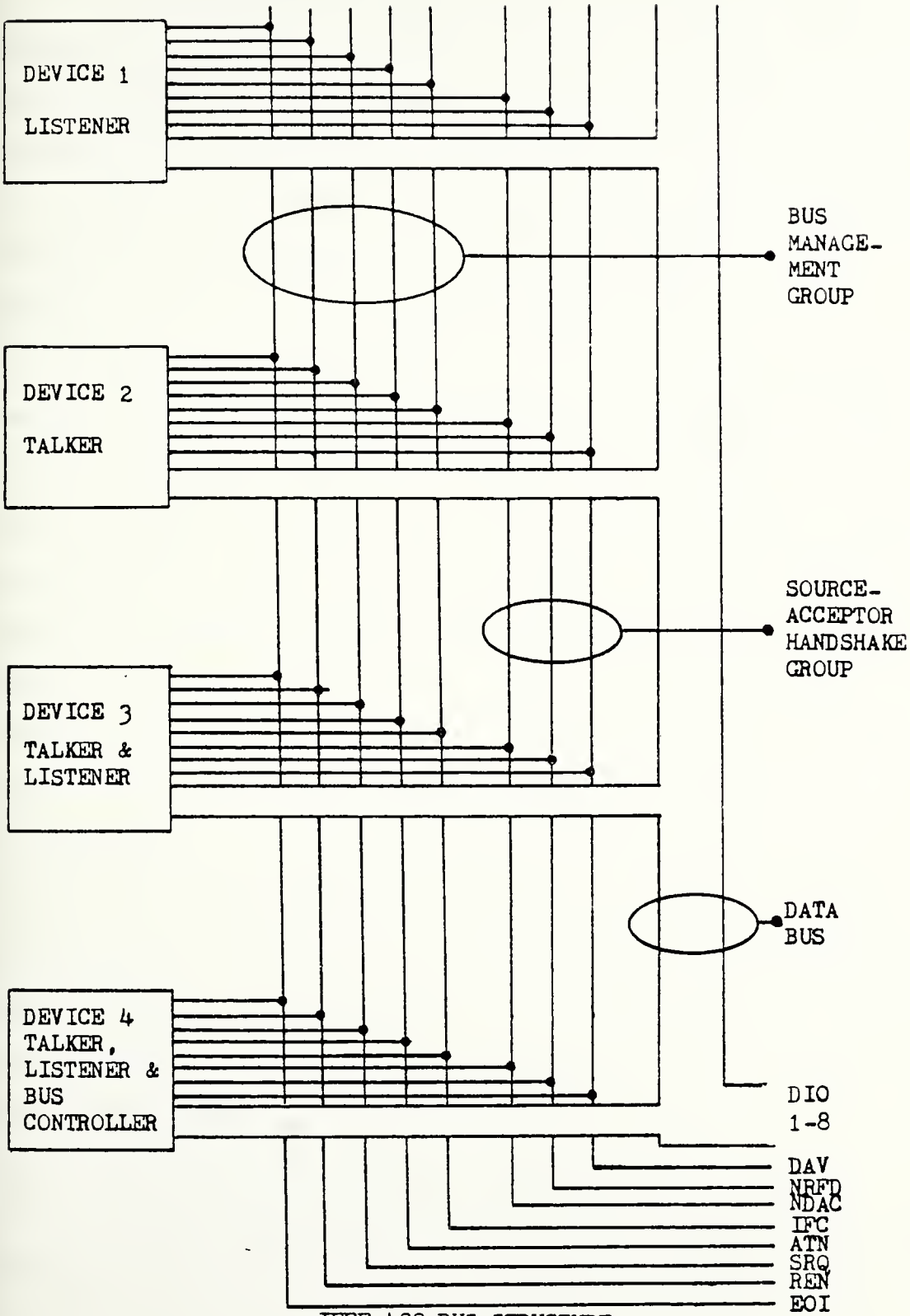
A. GENERAL

This appendix is provided as a ready reference for the reader who is unfamiliar with the IEEE 488 interface bus. It will briefly explain the bus structure, interface functions, and message encoding. The definitions provided for the signal lines and interface functions come directly from the IEEE standard to avoid any ambiguity in translation.

B. BUS STRUCTURE

The interface bus transfers data and control messages through 16 signal lines. The 16 lines are partitioned into three groups -- three lines for source-acceptor handshake protocol, five lines for bus management, and eight lines for bit parallel-byte serial transfer of device dependent data and some interface dependent messages.

Figure 16 shows a typical bus configuration. Bus configurations more complex than a single talker and one group of listeners will require a device to function as a bus controller. The bus controller is responsible for maintaining an orderly flow of both interface and device dependent messages. It must monitor the bus management lines and send appropriate address commands to talkers and listeners. Also, if more than one connected device can function as a bus controller, the controller-in-charge must guarantee that proper protocol is maintained to pass control from one device to another.



IEEE 488 BUS STRUCTURE
FIGURE 16

1. Source-Acceptor Handshake

The three line source-acceptor handshake group allows asynchronous transfer of data and multi-line messages from a device functioning as a talker to one or more devices functioning as listeners. The talker controls the logic level to the Data Valid (DAV) line, and the listeners collectively control the logic level of both the Not Ready for Data (NRFD) and the Not Data Accepted (NDAC) lines.

Because of the collective control of some of the bus management and source-acceptor handshake signal lines, active and passive message transfer is implemented. An active message transfer is one in which the message received is the message sent. A passive message transfer is one in which the message received is not necessarily the message sent. An example is the RFD (ready for data) message, a uniline message sent through the NRFD signal line. A listener will send the RFD message actively false until it becomes ready to receive data from the data bus; it will then send the RFD message passively true. This convention guarantees that the talker will not receive the RFD message true until all listeners are ready. Many of the other interface messages require this convention of active and passive message transfer.

The DAV line is used to indicate the condition (availability and validity) of information on the DIO signal lines.

The NRFD line is used to indicate the condition of readiness of device(s) to accept data.

The NDAC line is used to indicate the condition of acceptance of data by devices.

Figure 17 shows a typical timing diagram of the source-acceptor handshake lines for a multi-byte transfer of data from one talker to several listeners. The IEEE 488 standard used negative logic. A true message, logic 1, is represented electrically with a low voltage level ($\leq$.8V) The false message, logic 0, is a high voltage level ($\geq$2.0V).

2. Bus Management

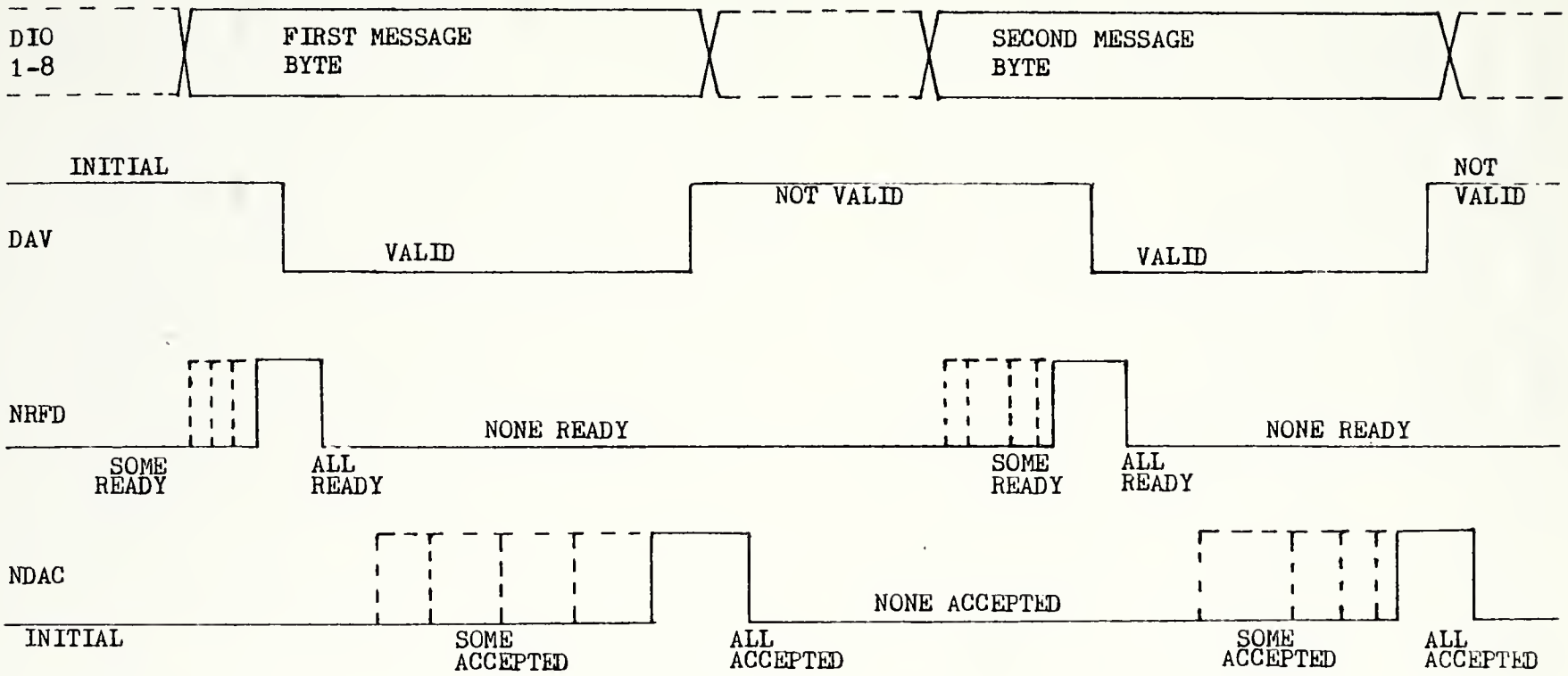
The five line bus management group allows the bus controller to monitor and control the interface bus. Each of the five lines has a specific function for the transfer of either uniline or multiline messages.

The Interface Clear (IFC) line is used to place the interface system, portions of which are contained in all interconnected devices, in a known quiescent state.

The Attention (ATN) line is used to specify how data on the DIO signal lines are to be interpreted and which devices must respond to the data.

The Service Request (SRQ) line is used by a device to indicate the need for attention and to request an interruption of the current sequence of events.

The Remote Enable (REN) line is used in conjunction with other messages to select between two alternate sources of device programming data.



SOURCE-ACCEPTOR TIMING SEQUENCE
FIGURE 17

The End or Identify (EOI) line is used to indicate the end of a multiple byte transfer or, in conjunction with ATN, to execute a polling sequence.

3. Data Transfer

The eight data lines are used to transfer device dependent messages from a talker to a listener. They are also used by the bus controller to pass address and universal commands.

In conjunction with the ATN line, the controller can pass talk and listen addresses to active devices. By using both the ATN and EOI lines, the controller can receive information as to the configuration or state of devices from the data lines.

C. INTERFACE FUNCTIONS

Interface functions are a part of every device designed to be attached to the interface bus. The functions provide the operational capability to send, receive, and process messages. Each function is assigned to process a fixed, but limited set of the total messages possible.

Depending on a specific application, a particular device may have only a few of the possible ten interface functions implemented. Additionally, the designer has the option to only implement a subset of the possible states within a particular function. The IEEE standard details each function, its states, and acceptable subsets to achieve bus compatibility.

Figure 18 shows the functional partition of interface function and message routing within a device. Figure 19 is an example of a state diagram for the SH function.

1. Source Handshake (SH)

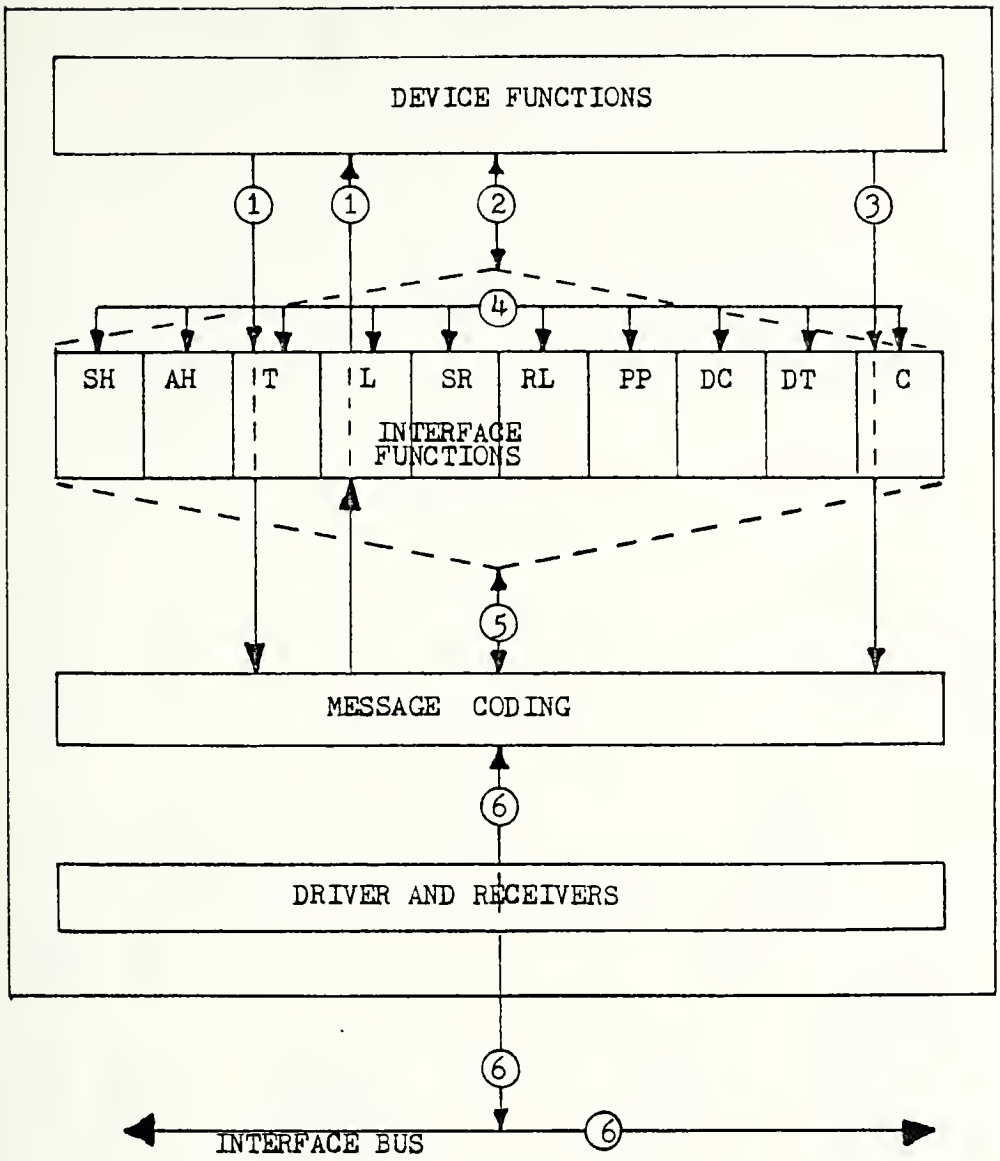
The SH interface function provides a device with the capability to guarantee the proper transfer of multiline messages. An interlocked handshake sequence between the SH interface function and one or more acceptor handshake functions (each contained within separate devices) guarantees asynchronous transfer of each multiline message. The SH interface function controls the initiation of and termination of the transfer of a multiline message byte. This function utilizes the DAV, RFD, and DAC messages for the message byte transfer.

2. Acceptor Handshake (AH)

The AH interface function provides a device with the capability to guarantee proper reception of remote multiline messages. An interlocked handshake sequence between an SH function and one or more AH functions guarantees asynchronous transfer of each message byte. An AH function may delay the initiation of, or the termination of, a multiline message transfer until prepared to continue with the transfer process. The AH function uses the DAV, RFD, and DAC messages for the message byte transfer.

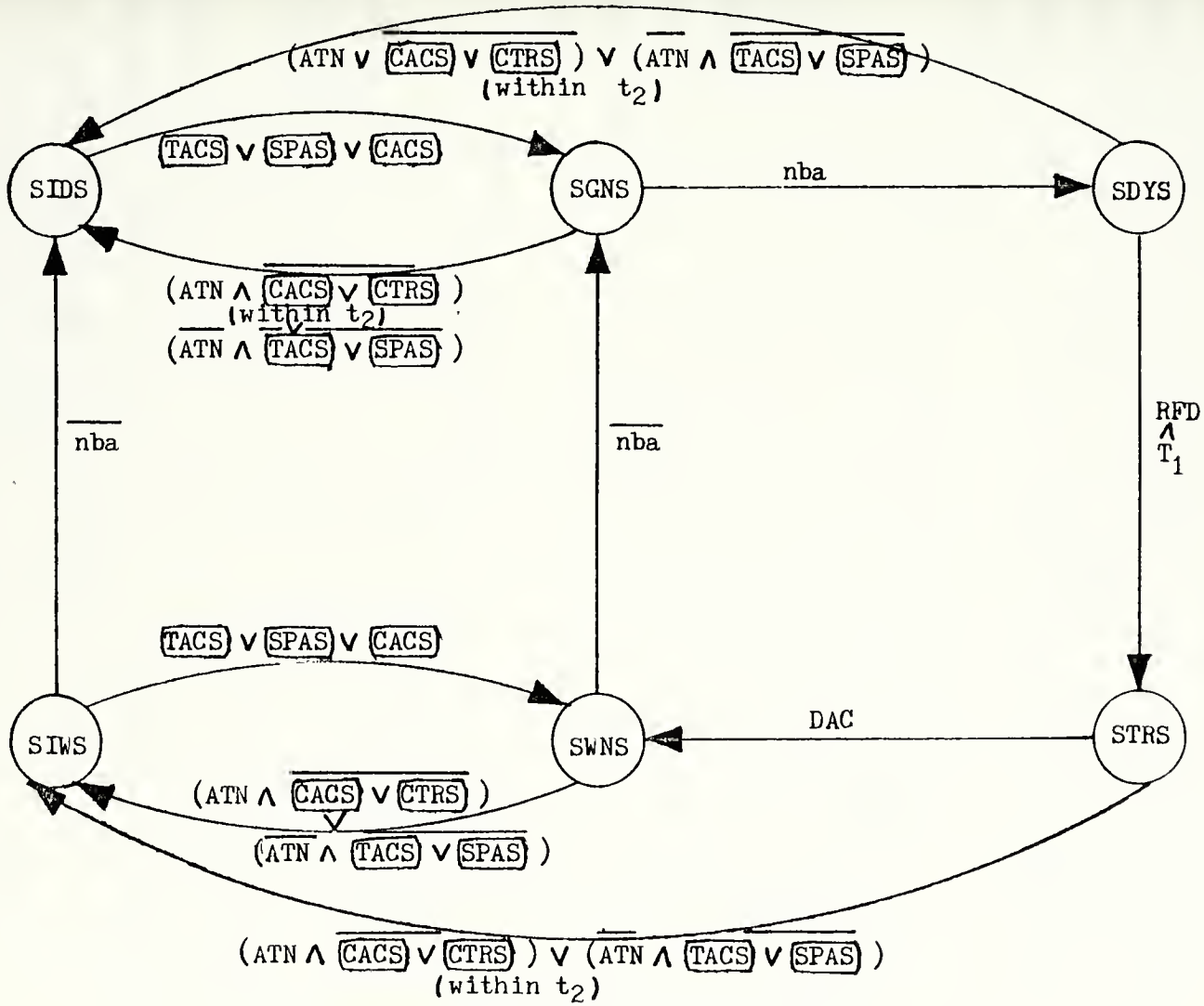
3. Talker (T) and Talker Extended (TE)

The T interface function provides a device with the capability to send device dependent data (including status



1. Device dependent messages to and from device functions
2. Local messages between interface functions and device functions
3. Interface messages sent by device function within a controller device
4. State linkage messages between interface functions
5. Remote interface messages between interface functions
6. Interface bus messages

IEEE 488 MESSAGE ROUTING
Figure 18



SH INTERFACE FUNCTION STATE DIAGRAM
Figure 19

data) over the interface to other devices. This capability exists only when the T interface function is addressed to talk. There are two alternative versions of the function: one with and one without address extension. The normal T function uses a one byte address. The T interface function with extended addressing (TE) uses a two byte address. Only one of the two alternative functions need be implemented in a specific device.

4. Listener (L) and Listener Extended (LE)

The L interface function provides a device with the capability to receive device dependent data (including status data) over the interface from other devices. This capability only exists when the device is addressed to listen. (As with the T function, the L function has extended addressing (LE).)

5. Service Request (SR)

The SR interface function provides a device with the capability to asynchronously request service from the controller in charge of the interface.

It also synchronizes the value of the service request bits of the status byte during a serial poll so that the SRQ message can be removed from the interface once this bit is true by the controller in charge.

6. Remote Local (RL)

The RL interface function provides a device with the capability to select between two sources of input information. The function indicates to the device that either

information from the front panel controls (local) or corresponding input information from the interface (remote) is to be used.

7. Parallel Poll (PP)

The PP interface function provides a device with the capability to present one bit of status to the controller in charge without being previously addressed to talk.

8. Device Clear (DC)

The DC interface function provides the device with the capability to be cleared (initialized) either individually or as part of a group of devices. The group may be either a subset or all addressed devices in one system.

9. Device Trigger (DT)

The DT interface function provides the device with the capability to have its basic operation started either individually or as part of a group of devices. The group may be either a subset or all addressed devices in one system.

10. Controller (C)

The C interface function provides a device with the capability to send device addresses, universal commands and addressed commands to other devices over the interface. It also provides the capability to conduct parallel polls to determine which devices require service. A C interface function can only exercise when it is sending the ATN message over the interface.

D. MESSAGES

Within a device, messages are categorized as either remote or local. Figure 18 illustrates the various routing paths for messages.

1. Local Messages

A local message is a message sent between a device function and an interface function. These messages are used to cause state transitions within the interface function when certain events have occurred. An example of a local message is the nba (new byte available) message required for the SH interface function. The nba message causes the transition from the SGNS (Source Generate) state to the SDYS (Source Delay) state. This is illustrated in Figure 19.

2. Remote Messages

A remote message is a message sent between interface functions of different devices via the instrument bus. Remote messages can be further categorized into interface messages and device dependent messages.

Interface messages are used to control the operation and configuration of the bus by causing state transition within the various interface functions. Figure 19 illustrates the transition from the SDYS state to the STRS (Source Transfer) state when the RFD message is received true.

Device dependent messages are transmitted via the instrument bus when the ATN message is false.

Device dependent messages do not cause state transition within interface functions. The IEEE standard places

no restrictions on the coding of these messages. The designer may use any mutually compatible coding between the interconnected devices. Examples of device dependent messages are device programming data, device measurement data, and device status data.

APPENDIX B

COMPUTER PROGRAMS

A. GENERAL

This appendix contains computer programs written for the testing and evaluation of the DFDR/M prototype. Two of the program modules contain utility routines which form the basis of procedures needed to program the MBM controller and the MC68488 General Purpose Interface Adapter. Additionally, routines are included to aid I/O operations and convert internal binary numbers to ASCII hexadecimal representation to aid in displaying the information on the communications unit attached to the SBC 80/20.

Two programs are application programs. These were specifically written to test the SBC 80/20 to MBM and the SBC 80/20 to IEEE 488 data bus interfaces. The RDWR program module tests for errors in the read/write of data to the magnetic bubble memory unit. Any errors which occur in this process are displayed on the console unit.

The GPIATS program module was written to test the interrupt feature of the MC68488 GPIA. When ASCII character strings were received from a HP-9825A calculator, functioning as the bus controller and talker, the characters were written to the console unit for verification.

The last program listed in this appendix is the HEXMEM program. This program converted a hexadecimal formatted file

[Ref. 12] into binary. The program was written to aid in the transfer of object code on a diskette file to the SBC 80/20 for PROM programming.

Four diskettes were used for file storage. A directory for each is included in Appendix C to explain the contents of each diskette.

PROGRAM: PRIMITIVE

DESCRIPTION: This was an utility program module. It contained procedures for I/O operations to the console device and for programming both the interrupt controller on the SBC 80/20 and the magnetic bubble memory controller.

All of the procedures were declared PUBLIC to allow their usage in other program modules. The object module was placed in a special library file (AIRDAT.LIB) created for the DFDR/M prototype. If modifications are made to this module, the library can be updated using the ISIS-II Library Manager [Ref. 12].

LIMITATIONS: None.

RECOMMENDED MODIFICATION: The I/O routines could be relocated in a separate module just for functional separation between modules.

A procedure to initialize the programmable USART on the SBC 80/20 should be added to eliminate the necessity to use the SBC 80/20 Monitor for that initialization.

If available memory is critical in a particular application, a significant saving in memory can be realized by not using the procedures to send or receive a value. The variable declarations at the beginning of the module memory-map instructions to the MBM controller.

Caution: Be sure to complement the data transferred from the CPU of the SBC 80/20 to the external devices because the data bus drivers invert the data.

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE PRIMITIVE
OBJECT MODULE PLACED IN :F1:PRIM.OBJ
COMPILER INVOKED BY:  PLM80 :F1:PRIM.PLM PAGELENGTH(35) PAGEWIDTH(80)


   1            PRIMITIVE:
                DO;
   2    1       DECLARE DCL LITERALLY 'DECLARE';
   3    1       DECLARE LIT LITERALLY 'LITERALLY';
   4    1       DCL TRUE LIT '0FFH',FALSE LIT '000H';
   5    1       DCL FOREVER LIT 'WHILE TRUE';

   6    1       DCL DATA$TO$FIFO BYTE AT(0FFF4H);
   7    1       DCL DATA$FROM$FIFO BYTE AT (0FFF3H);
   8    1       DCL PAGE$SEL$LO BYTE AT (0FFF0H);
   9    1       DCL PAGE$SEL$HI BYTE AT (0FFF1H);
  10    1       DCL CONT$COMMDS BYTE AT (0FFF2H);
  11    1       DCL BUBBLE$STATUS BYTE AT (0FFF5H);
  12    1       DCL PAGE$CNT$LO BYTE AT (0FFF6H);
  13    1       DCL PAGE$CNT$HI BYTE AT (0FFF7H);
  14    1       DCL MINOR$LOOP$SZLO BYTE AT (0FFF8H);
  15    1       DCL MINOR$LOOP$SZHI BYTE AT (0FFF9H);
  16    1       DCL PAGE$POS$LO BYTE AT (0FFFAH);
  17    1       DCL PAGE$POS$HI BYTE AT (0FFFBH);
  18    1       DCL PAGE$SIZE$REG BYTE AT (0FFFCH);
  19    1       DCL ROM$BYTE BYTE PUBLIC AT (0FFFDH);
  20    1       DCL ROM$ADDRS BYTE PUBLIC AT (0FFFEH);

                  /* THE FOLLOWING ROUTINES ARE TO COMMUNICATE TO A TERMINAL THAT
```

```
                    HAS BEEN INITIALIZED USING THE SBC MONITOR */

 21   1        IOINSTAT: PROCEDURE BYTE PUBLIC;
                     /* INPUT PORT 'ED' IS USART STATUS PORT */

 22   2            IF (INPUT(0EDH) AND 02H) > 0
                      THEN RETURN TRUE;
 24   2                 ELSE RETURN FALSE;

 25   2        END IOINSTAT;

 26   1        IOINDATA: PROCEDURE BYTE PUBLIC;
                     /* INPUT PORT 'EC' IS USART DATA PORT */

 27   2            RETURN INPUT(0ECH);

 28   2        END IOINDATA;

 29   1        IOOUTSTAT:  PROCEDURE BYTE PUBLIC;
                    /* INPUT PORT 'ED' IS USART STATUS PORT */

 30   2            IF (INPUT(0EDH) AND 01H) > 0
                      THEN RETURN TRUE;
 32   2                 ELSE RETURN FALSE;

 33   2        END IOOUTSTAT;

 34   1        CONO: PROCEDURE(CHAR) PUBLIC;
                    /* WRITE A CHARACTER TO THE CONSOLE */

 35   2            DCL CHAR BYTE;
```

```
 36   2          DO WHILE NOT IOOUTSTAT;
 37   3            END;    /* IDLE UNTIL USART READY TO TRANSMIT */

 38   2          OUTPUT(0ECH) = CHAR;

 39   2      END CONO;

 40   1      CONI: PROCEDURE BYTE PUBLIC;
                  /* READ A CHARACTER FORM CONSOLE WITH ECHO */

 41   2          DCL C BYTE;

 42   2          DO WHILE NOT IOINSTAT;
 43   3            END;
 44   2          C = IOINDATA;
 45   2          CALL CONO(C);
 46   2          RETURN C;


 47   2      END CONI;


 48   1      WRITE$STRING:  PROCEDURE(POINTER) PUBLIC;
                  /* THIS PROCEDURE IS TO WRITE A STRING OF CHARACTERS
                  LOCATED AT ADDRESS POINTER UNTIL A '*' IS
                  ENCOUNTERED */

 49   2          DCL POINTER ADDRESS;
 50   2          DCL CHAR BASED POINTER BYTE;

 51   2          DO WHILE CHAR <> '*';
 52   3              CALL CONO(CHAR);
```

```
 53   3              POINTER = POINTER + 1;
 54   3          END;

 55   2      END WRITE$STRING;

 56   1      CRLF: PROCEDURE PUBLIC;

 57   2          CALL WRITE$STRING(.(0DH,0AH,'*'));

 58   2       END CRLF;

              /*  THE FOLLOWING VARIABLES AND PROCEDURES ARE USED
                  IN CONJUNCTION WITH THE OPERATION OF THE MAGNETIC
                  BUBBLE MEMORY CONTROLLER.
                    DETAILS ON THE OPERATION OF THE MBM CONTROLLER ARE
                  FOUND IN TEXAS INSTRUMENTS TMS 9916 BUBBLE MEMORY
                  CONTROLLER MANUAL.    /*

              /*  NOTE:  THROUGHOUT THE FOLLOWING PROCEDURES ALL DATA
                  BEING SENT TO THE CONTROLLER MUST BE INVERTED TO ALLOW
                  FOR THE INVERSION THAT OCCURS AS DATA IS
                  TRANSMITTED ACROSS THE SYSTEMS BUS.   */

 59   1      READ$FIFO:  PROCEDURE BYTE PUBLIC;

                  /*  THIS PROCEDURE IS USED TO READ A BYTE
                       FROM THE MBM CONTROLLER'S FIFO */

 60   2          RETURN NOT DATA$FROM$FIFO;

 61   2      END READ$FIFO;
```

```
 62   1       WRITE$FIFO:  PROCEDURE(CHAR) PUBLIC;
                  /*  THIS PROCEDURE IS USED TO WRITE A BYTE OF DATA
                   TO THE MBM CONTROLLER'S FIFO  */

 63   2           DCL CHAR BYTE;

 64   2           DATA$TO$FIFO = NOT CHAR;

 65   2       END WRITE$FIFO;

 66   1       PAGE$READ$WRITTEN:  PROCEDURE BYTE PUBLIC;
                  /* THIS PROCEDURE IS USED TO RETURN STATUS
                  AS TO THE LAST COMMAND OF A READ OR WRITE OF A PAGE
                  OF DATA   */

 67   2           IF (NOT BUBBLE$STATUS AND 01H) > 0
                      THEN RETURN TRUE;
 69   2                ELSE RETURN FALSE;

 70   2       END PAGE$READ$WRITTEN;

 71   1       ADDRESSED$PAGE$AVAILABLE:  PROCEDURE BYTE PUBLIC;

                  /*  RETURNS  THE STATUS OF THE REQUESTED
                  PAGE */

 72   2           IF (NOT BUBBLE$STATUS AND 10H) > 0
                       THEN RETURN TRUE;
 74   2                 ELSE RETURN FALSE;

 75   2       END ADDRESSED$PAGE$AVAILABLE;
```

```
76   1       PAGE$COUNT$ZERO:  PROCEDURE BYTE PUBLIC;
                 /*  THIS PROCEDURE IS USED TO RETURN STATUS INFORMATION
                 FROM THE MBM CONTROLLER   */

77   2           IF (NOT BUBBLE$STATUS AND 40H) > 0
                     THEN RETURN TRUE;
79   2                ELSE RETURN FALSE;

80   2       END PAGE$COUNT$ZERO;

81   1       BYTE$READ$WRITTEN:  PROCEDURE BYTE PUBLIC;
                 /*  THIS PROCEDUE IS USED TO RETURN STATUS IN THE MULTI-
                 PAGE MODE TO INDICATE THAT A BYTE OF DATA
                 IS AVAILABLE IN THE FIFO.   */

82   2           IF (NOT BUBBLE$STATUS AND 80H) > 0
                     THEN RETURN TRUE;
84   2                ELSE RETURN FALSE;

85   2       END BYTE$READ$WRITTEN;

86   1       BUBBLE$BUSY:  PROCEDURE BYTE PUBLIC;

87   2           DCL I BYTE;

                 /*  THIS PROCEDURE IS USED TO RETURN THE OPERATIONAL
                STATUS OF THE MBM CONTROLLER.   IF IT IS BUSY
                WRITING INFORMATION TO FIFO, CAUTION MUST
                BE USED IN ISSUING OTHER COMMANDS */

88   2           DO I = 1 TO 2;
89   3             END;   /* THIS DO LOOP IS TO ASSURE THE BUBBLE
```

89

```
                        STATUS IS READY TO BE READ BEFORE CHECKING IT.
                        THERE IS AS MUCH AS A 10 MICRO SECOND DELAY
                        FROM THE READ OR WRITE PAGE COMMAND  */

 90   2            IF (NOT BUBBLE$STATUS AND 20H) > 0
                      THEN RETURN TRUE;
 92   2                 ELSE RETURN FALSE;

 93   2       END BUBBLE$BUSY;

 94   1       BUBBLE$RESET:  PROCEDURE PUBLIC;
                   /* THIS PROCEDURE IS USED TO ISSUE THE RESET
                   COMMAND TO THE MBM CONTROLLER.  ITS FUNCTION IS
                   DEFINED IN THE TMS 9916 CONTROLLER MANUAL */

 95   2            CONT$COMMDS = NOT 40H;

 96   2       END BUBBLE$RESET;

 97   1       INTERRUPT$MASK:  PROCEDURE PUBLIC;
                  /*  THIS PROCEDURE IS USED TO MASK THE INTERRUPT FROM THE
                    THE MBM CONTROLLER.  NOTE:  IT DOES NOT PREVENT THE
                    INTERRUPT PULSE FROM OCCURRING AT THE INTERRUPT
                    PIN.    */

 98   2            CONT$COMMDS = NOT 80H;

 99   2       END INTERRUPT$MASK;

100   1       SINGLE$PAGE: PROCEDURE PUBLIC;
                   /*  THIS PROCEDURE IS USED TO SELECT THE SINGLE
                   PAGE MODE IN THE MBM CONTROLLER */
```

```
101   2           CONT$COMMDS = NOT 08H;

102   2       END SINGLE$PAGE;

103   1       MULTI$PAGE:  PROCEDURE PUBLIC;
                  /* THIS PROCEDURE IS USED TO SELECT THE MULTIPAGE
                  MODE IN THE MBM CONTROLLER  */

104   2           CONT$COMMDS = NOT 10H;

105   2       END MULTI$PAGE;

106   1       READ$PAGE:  PROCEDURE PUBLIC;
                  /*  THIS PROCEDURE IS USED TO COMMAND THE MBM CONTROLLER
                  TO READ A PAGE OF DATA FROM THE MBM  INTO ITS
                  FIFO STACK.   */

107   2           CONT$COMMDS = NOT 02H;

108   2       END READ$PAGE;

109   1       WRITE$PAGE:  PROCEDURE PUBLIC;
                  /*  THIS PROCEDURE IS USED TO COMMMAND THE MBM CONTROLLER
                  TO WRITE THE PAGE OF DATA IN THE FIFO STACK TO THE
                  MBM    */

110   2           CONT$COMMDS = NOT 04H;

111   2       END WRITE$PAGE;

112   1       BUBBLECONT$INIT: PROCEDURE (PAGESIZE,MINORLOOPSIZE) PUBLIC;
```

```
                    /* THIS PROCEDURE IS USED TO INITIALIZE THE MBM CONTROLLER
                    USING THE PASSED PAGESIZE AND MINORLOOPSIZE.  */

113   2           DCL PAGESIZE BYTE;
114   2           DCL MINORLOOPSIZE ADDRESS;

115   2            CALL BUBBLE$RESET;
116   2            CALL INTERRUPT$MASK;

117   2            PAGE$SIZE$REG = NOT PAGESIZE;
118   2            MINOR$LOOP$SZLO = NOT LOW(MINORLOOPSIZE);
119   2            MINOR$LOOP$SZHI = NOT HIGH(MINORLOOPSIZE);

120   2            CONT$COMMDS = NOT 01H; /* INITIALIZE COMMAND*/
121   2            DO WHILE BUBBLE$BUSY; /* WAIT UNTIL BUBBLE FINISHED WITH
                                        THE INITIALIZATION BEFORE RETURNING */
122   3              END;

123   2       END BUBBLECONT$INIT;

124   1       SELECTPAGE:  PROCEDURE (PAGENUM) PUBLIC;
                /*  THIS PROCEDURE IS USED TO LOAD THE PASSED PAGE
                NUMBER INTO THE PAGE  SELECT REGISTERS */

125   2            DCL PAGENUM ADDRESS;

126   2            PAGE$SEL$LO = NOT LOW(PAGENUM);
127   2            PAGE$SEL$HI = NOT HIGH(PAGENUM);

128   2       END SELECTPAGE;

129   1       LOAD$PAGE$COUNT:  PROCEDURE(PAGECOUNT) PUBLIC;
```

```
                    /* THIS PROCEDURE IS USED TO LOAD THE NUMBER
                    OF PAGES TO BE TRANSFERRED IN THE MULTIPAGE MODE */

130   2             DCL PAGECOUNT ADDRESS;

131   2             PAGE$CNT$LO = NOT LOW(PAGECOUNT);
132   2             PAGE$CNT$HI = NOT HIGH(PAGECOUNT);

133   2         END LOAD$PAGE$COUNT;

134   1         LOAD$PAGE$POSITION:  PROCEDURE(PAGEPOSITION) PUBLIC;
                    /*  THIS PROCEDURE IS USED TO LOAD THE PAGE POSITION
                    REGISTER.  WOULD BE USED TO CHANGE THE NEUTRAL POSITION
                   OF THE MBM  */


135   2             DCL PAGEPOSITION ADDRESS;

136   2             PAGE$POS$LO = NOT LOW(PAGEPOSITION);
137   2             PAGE$POS$HI = NOT HIGH(PAGEPOSITION);

138   2         END LOAD$PAGE$POSITION;

139   1         READ$PAGE$COUNT:  PROCEDURE ADDRESS PUBLIC;
                    /* THIS PROCEDURE IS USED TO RETURN THE CURRENT VALUE
                    IN THE PAGE COUNT REGISTER OF THE MBM CONTROLLER */

140   2             DCL PAGEHIGH ADDRESS;

141   2             PAGEHIGH = SHL(DOUBLE(NOT PAGE$CNT$HI),8);
142   2             RETURN PAGEHIGH + (NOT PAGE$CNT$LO);
```

```
143   2       END READ$PAGE$COUNT;

144   1       READ$PAGE$POSITION: PROCEDURE ADDRESS PUBLIC;
                  /*  THIS PROCEDURE IS USED TO RETURN THE CURRENT PAGE POSITION
                  NOTE:  THIS CAN BE READ DURING THE BUBBLE$BUSY FOR STATUS
                  INFORMATION.  AT COMPLETION OF A PAGE READ OR WRITE
                  THIS REGISTER WILL NORMALLY BE ZERO  */

145   2           DCL PAGEHIGH ADDRESS;
146   2           PAGEHIGH = SHL(DOUBLE(NOT PAGE$POS$HI),8);
147   2           RETURN PAGEHIGH + NOT PAGE$POS$LO;

148   2       END READ$PAGE$POSITION;


149   1       HEXTOASCII: PROCEDURE(CONVERS) PUBLIC;
150   2           DCL CONVERS ADDRESS;
151   2           DCL CONVERHI BYTE;
152   2           DCL CONVERLO BYTE;
153   2           DCL ASCIIOUT (5) BYTE;
154   2           DCL I BYTE;

155   2           DO;
156   3               CONVERHI = HIGH(CONVERS);
157   3               CONVERLO = LOW(CONVERS);
158   3               ASCIIOUT(0) = SHR(CONVERHI,4);
159   3               ASCIIOUT(1) = CONVERHI AND 0FH;
160   3               ASCIIOUT(2) = SHR(CONVERLO,4);
161   3               ASCIIOUT(3)= CONVERLO AND 0FH;

162   3               DO I = 0 TO 3;
163   4                   IF ASCIIOUT(I) > 9
```

```
                                THEN
164    4                        ASCIIOUT(I) = ASCIIOUT(I) - 9 + 40H;
                                ELSE
165    4                        ASCIIOUT(I) = ASCIIOUT(I) + 30H;
166    4                    END;

167    3                ASCIIOUT(4) = '*';
168    3                CALL WRITE$STRING(.ASCIIOUT);
169    3            END;

170    2        END HEXTOASCII;


171    1        PRO8259:  PROCEDURE(I40R8,INTVECTOR) PUBLIC;
                    /* THIS PROCEDURE IS USED TO SEND THE INITIALIZATION
                    COMMAND WORDS TO THE PROGRAMMABLE INTERRUPT CON-
                    TROLLER.   IT RECEIVES TWO PARAMETERS:
                      I40R8 MUST BE  4 OR 8;
                        IF OTHER THAN THESE A DEFAULT OF 8
                        WILL BE USED.
                      INTVECTOR IS THE BEGINNING ADDRESS OF THE INTER-
                         RUPT VECTOR.   FOR I40R8 = 4 IT MUST BE A MULTI-
                         BLE OF 20H,   FOR I40R8 = 8 IT MUST BE A MULTI-
                         BLE OF 40H.    */

172    2            DCL I40R8 BYTE;
173    2            DCL INTVECTOR ADDRESS;

174    2            IF I40R8 = 4
                         THEN OUTPUT(0D8H) = 16H +(LOW(INTVECTOR) AND 0E0H);
176    2                    ELSE OUTPUT(0D8H) = 12H +(LOW(INTVECTOR)AND 0C0H);
177    2            OUTPUT(0D9H) = HIGH(INTVECTOR);
```

```
 178   2       END PRO8259;

 179   1       PIC8259$MASK: PROCEDURE(INTMASK) PUBLIC;
                     /* THIS PROCEDURE IS USED TO MASK INTERRUPT FROM THE CPU
                     BASED ON THE VALUE OF INTMASK  */

 180   2           DCL INTMASK BYTE;

 181   2           OUTPUT(0D9H) = INTMASK;

 182   2       END PIC8259$MASK;
 183   1       END PRIMITIVE;
```


MODULE INFORMATION:

```
     CODE AREA SIZE      = 029DH    669D
     VARIABLE AREA SIZE  = 0021H     33D
     MAXIMUM STACK SIZE  = 0006H      6D
     391 LINES READ
     0 PROGRAM ERROR(S)
```

END OF PL/M-80 COMPILATION

PROGRAM: GPIAMOD

DESCRIPTION: This was a utility program module. It contains procedures to control and monitor the MC68488 GPIA. Each of the procedures interacts with a particular register and has a specific function. All of the procedures were declared PUBLIC to allow their usage in other program modules.

The object code file was placed in a special library (AIRDAT.LIB) created for use with programs written for the DFDR/M prototype. If modifications are made to this program module, the library file can be updated using the ISIS-II Library Manager [Ref. 12].

LIMITATIONS: None.

SUGGESTED MODIFICATIONS: If available memory is critical in a particular application, a significant saving of space can be realized by not using the procedures contained herein to send or return a value from the GPIA. Use of the variable declarations at the beginning of the module is recommended.

Caution: Be sure to complement the data transferred from the CPU of the SBC 80/20 to the external devices because the data bus drivers invert the data.

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE GPIAMOD
OBJECT MODULE PLACED IN :F1:GPIA.OBJ
COMPILER INVOKED BY:  PLM80 :F1:GPIA.PLM PAGELENGTH(35) PAGEWIDTH(80)
```

```
                /* THIS PLM MODULE WILL CONTAIN THE NECESSARY PRIMITIVES
                TO CONTROL THE OPERATION OF THE MC 68488 GENERAL
                PURPOSE INTERFACE ADAPTER(GPIA).  THIS ADAPTER
                ALLOWS THE SBC 80/20 TO COMMUNICATE WITH THE IEEE 488-1975
                DATA BUS THROUGH WHICH THE SYSTEM WILL RECEIVE DIGITAL FLIGHT
                DATA.  NOTE:  IN ALL OF THE SUBROUTINES IT IS NECESSARY TO
                COMPLEMENT THE DATA GOING ACROSS THE SYSTEMS BUS.  THIS IS
                BECAUSE OF THE INVERTING BUS DRIVERS.    */

   1            GPIAMOD: DO;

   2    1       DECLARE DCL LITERALLY 'DECLARE';
   3    1       DECLARE LIT LITERALLY 'LITERALLY';

                /* THE FOLLOWING ARE DATA VARIABLES TO ALLOW ACCESS TO THE
                VARIOUS WORKING REGISTERS OF THE GPIA.  THE DEVICE IS
                MEMORY MAPPED TO LOCATIONS FFE0H TO FFE7H.  */

   4    1       DCL GPIA$INTMASKREG BYTE AT (0FFE0H);
   5    1       DCL GPIA$INTSTATREG BYTE AT (0FFE0H);
   6    1       DCL GPIA$COMMDSTATREG BYTE AT (0FFE1H);
   7    1       DCL GPIA$ADDRSTATREG BYTE AT (0FFE2H);
   8    1       DCL GPIA$ADDRMODEREG BYTE AT (0FFE2H);
   9    1       DCL GPIA$AUXCOMMDREG BYTE AT (0FFE3H);
```

```
10   1       DCL GPIA$ADDRSWITREG BYTE AT (0FFE4H);
11   1       DCL GPIA$ADDRREG BYTE AT (0FFE4H);
12   1       DCL GPIA$SERPOLLREG BYTE AT (0FFE5H);
13   1       DCL GPIA$COMMDPASSTHRU BYTE AT (0FFE6H);
14   1       DCL GPIA$PARLLPOLLREG BYTE AT (0FFE6H);
15   1       DCL GPIA$DATAINREG BYTE AT (0FFE7H);
16   1       DCL GPIA$DATAOUTREG BYTE AT (0FFE7H);

             /*  THE FOLLOWING ROUTINES ARE WRITTEN TO SIMPLIFY SYSTEM
             DESIGN AND IMPLEMENTATION.  IT IS REALIZED THAT IF MEMORY
             USEAGE BECOMES A FACTOR, MANY SHORT CUTS CAN BE TAKEN
             TO STREAM LINE THE MACHINE LANGUAGE CODE BY REDUCING THE NUMBER
             OF PROCEDURE CALLS.  */

17   1       GPIAINTERMASK: PROCEDURE(MASKBYTE) PUBLIC;

                 /* THE INTERRUPT MASK REGISTER IS AS FOLLOWS:
                   ---------------------------------------------
                  \ IRQ \ BO  \ GET \ XXX \ APT \ CMD \ END \ BI  \
                   ---------------------------------------------
                 IRQ : MASK BIT FOR THE IRQ PIN.  MUST BE HIGH FOR ANY
                       INTERRUPT TO BE PASSED TO THE MICRO-PROCCESSOR.
                 BO  : INTERRUPT WHEN BYTE IN DATA OUT REGISTER IS TRANSMITTED.
                 GET : INTERRUPT ON GROUP EXECUTE TRIGGER SIGNAL.
                 APT : INTERRUPT ON SECONDARY ADDRESS PASS-THRU
                 CMD : INTERRUPT ON SPAS + RLC +DSEL(DCAS+UUCG+UACG)
                 END : INTERUPT ON EOI AND ATN
                 BI  : INTERRUPT ON BYTE RECEIVED IN DATA IN REGISTER
                                                                    */

18   2           DCL MASKBYTE BYTE;
```

```
 19   2          GPIA$INTMASKREG = NOT MASKBYTE;

 20   2      END GPIAINTERMASK;

 21   1      GPIAINTERSTAT: PROCEDURE BYTE PUBLIC;
              /* THE BIT DEFINITIONS OF THIS REGISTER ARE IDENTICAL
              TO THOSE FOUND IN THE INTERRUPT MASK REGISTER EXCEPT
              BIT 7 IS INT.  THE INT BIT IS THE LOGICAL OR
              OF ALL OTHER BITS IN THIS REGISTER AND'ED WITH THE
              RESPECTIVE BITS IN THE INTERRUPT MASK REGISTER.
              IF INTERRUPTS ARE NOT MASKED, THIS REGISTER CAN BE USED
              TO DETERMINE WHAT ACTION CAUSED THE INTERRUPT.  NOTE:
              THIS REGISTER MUST BE READ TO RESET THE IRQ SIGNAL PIN.
              ADDITIONALLY, IF ALL INTERRUPTS HAVE BEEN MASKED, THIS
              REGISTER CAN BE USED TO POLL BECAUSE THE BITS ARE SET REGARD-
              LESS OF THE STATE OF THE MASK REGISTER.     */
 22   2          RETURN NOT GPIA$INTSTATREG;

 23   2      END GPIAINTERSTAT;

 24   1      GPIACOMMANDSTAT: PROCEDURE BYTE PUBLIC;
             /* THE RETURNED VALUE HAS THE FOLLOWING FORMAT:
              -------------------------------------------------
              \ UACG\ REM \ LOK \ XXX \ RLC \ SPAS\ DCAS\ UUCG\
              -------------------------------------------------
              UACG : UNDEFINED ADDRESSED COMMAND
              REM  : REMOTE ENABLED
              LOK  : LOCAL LOCKOUT ENABLED
              RLC  : REMOTE/LOCAL STATE CHANGED
              SPAS : SERIAL PLL ACTIVE STATE IS IN EFFECT
              DCAS : DEVICE CLEAR ACTIVE STATE IS IN EFFECT
              UUCG : UNDEFINED UNIVERSAL COMMAND
```

```
                THIS REGISTER CAN BE READ IF AN INTERRUPT IS CAUSED
                BY THE CMD INTERRUPT BIT   */

  25   2            RETURN NOT GPIA$COMMDSTATREG;

  26   2        END GPIACOMMANDSTAT;

  27   1        GPIAADDRESSSTAT: PROCEDURE BYTE PUBLIC;
                    /*  THE RETURNED VALUE HAS THE FOLLOWING FORMAT:
                    ------------------------------------------------
                    \ MA  \ TO  \ LO  \ ATN \ TACS\ LACS\ LPAS\ TPAS\
                    ------------------------------------------------

                    MA    : MY ADDRESS HAS OCCURED
                    TO    : TALK-ONLY MODE IS ENABLED
                    LO    : LISTEN-ONLY MODE IS ENABLED
                    ATN   : THE ATTENTION COMMAND IS ASSERTED
                    TACS  : THE GPIA IS IN THE TALKER ACTIVE STATE
                    LACS  : THE GPIA IS IN THE LISTEN ACTIVE STATE
                    LACS  : THE FPIA IS IN THE LISTENER PRIMARY
                            ADDRESSED STATE
                    TPAS  : THE GPIA IS IN THE TALKER PRIMARY
                            ADDRESSED STATE   */

  28   2            RETURN NOT GPIA$ADDRSTATREG;

  29   2        END GPIAADDRESSSTAT;

  30   1        GPIAADDRESSMODE: PROCEDURE(ADDRSMODE) PUBLIC;
                    /* THE ADDRESS MODE REGISTER HAS THE FOLLOWING
                       FORMAT:
```

80

```
                    ------------------------------------------------
                    \ DSEL\ TO  \ LO  \ XXX \ HDLE\ HDLA\ XXX \ APTE\
                    ------------------------------------------------
                    DSEL : CONFIGURE FOR AUTOMATIC COMPLETION OF HAND-
                           SHAKE SEQUENCE ON OCCURRENCE OF GET, UACG,
                           UUCG, SDC, OR DCL COMMANDS
                    TO   : ENTER TALKER-ONLY MODE
                    LO   : ENTER LISTENER-ONLY MODE
                    HDLE : HOLD-OFF RFD ON END
                    HDLA : HOLD-OFF RFD ON ALL DATA
                    APTE : ADDRESS PASS-THRU FEATURE ENABLE
                                                            */
 31   2             DCL ADDRSMODE BYTE;

 32   2             GPIA$ADDRMODEREG = NOT ADDRSMODE;

 33   2         END GPIAADDRESSMODE;

 34   1         SETDEVICEADDRESS: PROCEDURE(SPECIALMODE) PUBLIC;
                    /* FORMAT OF ADDRESS SWITCH REGISTER:
                    ------------------------------------------------
                    \ UD3 \ UD2 \ UD1 \ AD5 \ AD4 \ AD3 \ AD2 \ AD1 \
                    ------------------------------------------------
                    UD3 - UD1 : USER DEFINED BITS, USEFUL TO DECODE
                                DEVICE FUNCTION IF NEED BE
                    AD5 - AD1 : THE DEVICES ADDRESS READ FROM THE
                                ADDRESS SELECT SWITCHES

                       FORMAT OF ADDRESS REGISTER:
                    ------------------------------------------------
                    \ ISBE\ DAL \ DAT \ AD5 \ AD4 \ AD3 \ AD2 \ AD1 \
                    ------------------------------------------------
```

```
                    ISBE : ENABLE THE DUAL PRIMARY ADDRESSING MODE
                    DAL  : DISABLE THE LISTENER
                    DAT  : DISABLE THE TALKER     AD5 - AD1 : DEVICE ADDRESS AS DE
              -   FINED ABOVE

                    NOTE: THE USER DEFINED BITS ARE NOT USED ON THE
                          ADDRESS SELECT SWITCHES SINCE THE CURRENT
                          USE IS  LISTEN ONLY    */

  35    2           DCL SPECIALMODE BYTE;

  36    2           GPIA$ADDRREG =NOT ((NOT GPIA$ADDRSWITREG AND 1FH) OR
                                  (SPECIALMODE AND 0E0H));

  37    2       END SETDEVICEADDRESS;


  38    1       GPIAREADAUXCOMM: PROCEDURE BYTE PUBLIC;
                    /* THE FORMAT OF THE AUXILIARY COMMAND REGISTER IS:
                    ------------------------------------------------
                    \RESET\ DAC \ DAV \ RFE \ MSA \ RTL \ ULPA\ FGET\
                    ------------------------------------------------

                    RESET: CHIP IN RESET MODE
                    DAC  : INDICATES STATE OF THE DAC PIN
                    DAV  : INDICATES STATE OF THE DAV PIN
                    RFD  : INDICATES STATE OF THE DRFD PIN
                    MSA  : INDICATES IF MSA IS SET
                    RTL  : STATE OF RETURN TO LOCAL
                    ULPA : STAT OF LSB OF VUS AT LAST PRIMARY ADDRESS
                           RECEIVE TIME
                    FGET : FORCE GROUP TRIGGER COMMAND FROM MICRO-PROCESSOR
```

```
                        HAS OCCURED
                                                */

 39   2           RETURN NOT GPIA$AUXCOMMDREG;

 40   2       END GPIAREADAUXCOMM;

 41   1       GPIAWRITEAUXCOMM: PROCEDURE(COMMDIN) PUBLIC;
                   /*  THE FORMAT OF THE AUX COMMAND REG. IS:
                   ------------------------------------------------
                   \RESET\ RFDR\ FEOI\ DACR\ MSA \ RTL \ DACD\ FGET\
                   ------------------------------------------------
                   RESET: SET FOR SOFTWARE RESET
                   RFDR : COMPLETE HANDSHAKE STOPPED BY THE RFD HOLD-OFF
                   FEOI : SET EOI MESSAGE TRUE, CLEARED AFTER BYTE TRANS-
                          MITTED
                   DACR : SET TO INDICATE THAT THE MICRO-PROCESSOR HAS EXAMINED
                          AN UNDEFINED COMMAND OR SECONDARY ADDRESS
                   MSA  : SETTING WILL CAUSE THE GPIA TO TRANSFER INTO LADS
                          OR TADS IF IT WAS IN LPAS OR TDAS
                   RTL  : RETURN TO LOCAL IF LOCKOUT IS DISABLED
                   DACD : DISABLE AUTO HANDSHAKE ON ADDRESS OR
                           COMMANDS
                   FGET : FORCE GROUP EXECUTE TRIGGER
                                                   */
 42   2           DCL COMMDIN BYTE;

 43   2           GPIA$AUXCOMMDREG = NOT COMMDIN;

 44   2       END GPIAWRITEAUXCOMM;
```

```
45   1       GPIASERIALPOLL: PROCEDURE(RW,POLLSTATUS) BYTE PUBLIC;
                 /* THE FORMAT OF THE SERIAL POLL REGISTER IS:
                 ------------------------------------------------
                 \ S8  \ SRQS\ S6  \ S5  \ S4  \ S3  \ S2  \ S1  \
                 ------------------------------------------------
                 S8 - S1 : STATUS BITS
                 SRQS : INDICATES THAT THE BUS IS IN THE SERVICE
                        REQUEST STATE (A READ REGISTER)
                 NOTE:  THE SRQS BIT BECOMES RSV BIT ON A WRITE TO THE
                         SERIAL POLL REGISTER.
                                                          */
46   2           DCL (RW,POLLSTATUS) BYTE;

                  /* RW USED TO DETERMINE IF THE OPERATION IS A
                    READ OR A WRITE TO THE REGISTER.   */

47   2           IF RW = 0
                     THEN   /* READ OPERATION */
48   2                 RETURN NOT GPIA$SERPOLLREG;
                     ELSE   /* WRITE OPERATION */
49   2                 GPIA$SERPOLLREG = NOT POLLSTATUS;

50   2            RETURN 00;

51   2       END GPIASERIALPOLL;

52   1       GPIACOMMANDPASSTHRU: PROCEDURE BYTE PUBLIC;
                 /*  THE COMMAND-PASS-THRU REGISTER FORMAT IS:
                 ------------------------------------------------
                 \ B7  \ B6  \ B5  \ B4  \ B3  \ B2  \ B1  \ B0  \
                 ------------------------------------------------
```

```
                    B7 - B0 : PASS COMMANDS AND SECONDARY ADDRESS TO
                              MICRO-PROCESSOR
                                                              */

 53   2             RETURN NOT GPIA$COMMDPASSTHRU;

 54   2         END GPIACOMMANDPASSTHRU;

 55   1         GPIADATAIN: PROCEDURE BYTE PUBLIC;
                    /* THE DATAIN FORMAT IS AS FOLLOWS:
                    ------------------------------------------
                    \ DI7 \ DI6 \ DI5 \ DI4 \ DI3 \ DI2 \ DI1 \ DI0 \
                    ------------------------------------------
                    DI0 - DI7 : BITS CORRESPONDING TO DATA RECEIVED
                                FROM THE DATA BUS       */

 56   2             RETURN NOT GPIA$DATAINREG;

 57   2         END GPIADATAIN;

 58   1         GPIADATAOUT: PROCEDURE(DATABYTE) PUBLIC;
                    /* THE DATA OUT REGISTER FORMAT IS:
                    ------------------------------------------------
                    \ DO7 \ DO6 \ DO5 \ DO4 \ DO3 \ DO2 \ DO1 \ DO0 \
                    ------------------------------------------------
                    DO7 - DO0: BITS CORRESPONDING TO DATA TO BE
                                TRANSMITTED TO THE IEEE-488 DATABUS  */

 59   2             DCL DATABYTE BYTE;

 60   2             GPIA$DATAOUTREG = NOT DATABYTE;
```

```
 61    2       END GPIADATAOUT;
 62    1       END GPIAMOD;


MODULE INFORMATION:

     CODE AREA SIZE      = 0083H    131D
     VARIABLE AREA SIZE  = 0007H      7D
     MAXIMUM STACK SIZE  = 0002H      2D
     283 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

PROGRAM: RDWR

DESCRIPTION: The RDWR program was written to test the operation of the magnetic bubble memory module. It was designed to sequentially write a byte value beginning with 00H through 0FFH throughout the usable bubble memory. After the write of a particular byte value, a read was done to detect transfer errors.

If errors occurred, the page number, byte position, and error value would be written to the console.

LIMITATIONS: Program was compiled to be executed in the SBC 80/20's RAM. ICE-80 emulation was the only practical method to load this program.

The error rate during testing was considerably greater than anticipated. The result was an excessive listing of errors.

RECOMMENDED MODIFICATIONS: This program has a few limitations, but when the MBM write/read error rate is reduced, it will be a viable test. However, until that time, the program may be modified to only report the number of errors found on the write/read of each byte value.

The high error rate was caused by the MBM controller to MBM interface. The byte values returned from the MBM controller's FIFO were shifted left by one bit position. A modification to this program, or one similar to it, could be used to adjust the 18 byte FIFO contents back to the right. Such a fix is recommended until the exact cause for the

shifting left problem is determined and corrected in hardware.

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE RDWR
OBJECT MODULE PLACED IN :F1:RDWR.OBJ
COMPILER INVOKED BY:  PLM80 :F1:RDWR.PLM PAGELENGTH(35) PAGEWIDTH(80)



   1          RDWR: DO;

   2   1      DECLARE DCL LITERALLY 'DECLARE';
   3   1      DECLARE LIT LITERALLY 'LITERALLY';

   4   1      DCL (I,J,K) ADDRESS;
   5   1      DCL FIFOBYTE BYTE;

   6   1      BUBBLECONT$INIT: PROCEDURE(PAGESIZE,MINORLOOPSIZE) EXTERNAL;

   7   2          DCL PAGESIZE BYTE;
   8   2          DCL MINORLOOPSIZE ADDRESS;
   9   2       END BUBBLECONT$INIT;

  10   1      WRITE$STRING: PROCEDURE(POINTER) EXTERNAL;
  11   2          DCL POINTER ADDRESS;
  12   2       END WRITE$STRING;

  13   1      HEXTOASCII: PROCEDURE(CONVERS) EXTERNAL;
  14   2           DCL CONVERS ADDRESS;
  15   2       END HEXTOASCII;

  16   1      CRLF: PROCEDURE EXTERNAL;
  17   2       END CRLF;
```

```
18   1      BUBBLE$BUSY: PROCEDURE BYTE EXTERNAL;
19   2       END BUBBLE$BUSY;

20   1      WRITE$FIFO: PROCEDURE(CHAR) EXTERNAL;
21   2         DCL CHAR BYTE;
22   2       END WRITE$FIFO;

23   1      SELECTPAGE: PROCEDURE(PAGENUM) EXTERNAL;
24   2         DCL PAGENUM ADDRESS;
25   2       END SELECTPAGE;

26   1      READ$PAGE: PROCEDURE EXTERNAL;
27   2        END READ$PAGE;

28   1      WRITE$PAGE: PROCEDURE EXTERNAL;
29   2        END WRITE$PAGE;

30   1      READ$FIFO: PROCEDURE BYTE EXTERNAL;
31   2       END READ$FIFO;

32   1      PIC8259$MASK: PROCEDURE(INTMASK) EXTERNAL;
33   2         DCL INTMASK BYTE;
34   2       END PIC8259$MASK;

35   1      DO;
36   2      CALL PIC8259$MASK(20H);  /*MASK INTRP 5
                                       FROM BUBBLE CONTROLLER */

37   2      CALL BUBBLECONT$INIT(18,641);
38   2      DO I = 0 TO 255;
39   3          CALL WRITE$STRING(.('WRITE OF *'));
40   3          CALL HEXTOASCII(I);
```

```
41   3          CALL CRLF;
42   3          DO J = 0 TO 640;
43   4               DO WHILE BUBBLE$BUSY;
44   5               END;
45   4               DO K = 1 TO 18;
46   5                    CALL WRITE$FIFO(I);
47   5               END;
48   4               CALL SELECTPAGE(J);
49   4               CALL WRITE$PAGE;
50   4          END;


51   3          CALL WRITE$STRING(.('READ BEGINNING *'));
52   3          CALL CRLF;
53   3          DO WHILE BUBBLE$BUSY;
54   4          END;

55   3          DO J = 0 TO 640;
56   4               CALL SELECTPAGE(J);
57   4               CALL READ$PAGE;

58   4                DO WHILE BUBBLE$BUSY;
59   5                 END;

60   4                DO K = 1 TO 18;
61   5                     IF (FIFOBYTE := READ$FIFO) <> I
                             THEN
62   5                      DO;
63   6                      CALL WRITE$STRING(.('ERROR PAGE *'));
64   6                      CALL HEXTOASCII(J);
65   6                      CALL WRITE$STRING(.(' BYTE NO. *'));
66   6                      CALL HEXTOASCII(K);
```

```
  67   6                        CALL WRITE$STRING(.(' BYTE READ IS *'));
  68   6                        CALL HEXTOASCII(FIFOBYTE);
  69   6                        CALL CRLF;
  70   6                        END;
  71   5                    END;
  72   4                END;
  73   3            END;
  74   2            END;
  75   1            END RDWR;


MODULE INFORMATION:

     CODE AREA SIZE      = 017DH    381D
     VARIABLE AREA SIZE  = 0007H      7D
     MAXIMUM STACK SIZE  = 0002H      2D
     99 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

PROGRAM: GPIATS

DESCRIPTION: The GPIATS program was written for execution in the SBC 80/20. This program tested the interrupt feature of the MC68488 GPIA. After initialization of the GPIA and the SBC 80/20 interrupt controller, execution of a continuous loop begins until an interrupt from the GPIA occurs. The interrupt from the GPIA occurs when it has received a data byte from a talker on the IEEE 488 instrument data bus.

The interrupt service routine resets the GPIA's interrupt request pin, reads the ASCII data byte from the GPIA's data-in register, sends it to the console unit, and finally sends a non-specific End of Interrupt to the interrupt controller.

This program was placed in ROM. It was compiled to be executed at 0820H. The interrupt jump vector begins at 0800H. Data variables begin at 3000H and the stack at 3C1EH.

LIMITATIONS: The SBC 80/20's monitor program must be used to initialize the baud rate of the programmable USART. Execution is begun by using the monitor's GO command.

RECOMMENDED MODIFICATIONS: None. The operation of the GPIA interface was demonstrated. A version of the Interrupt 6 procedure contained in this thesis may be used in the operational program when written.

Caution must be used in coding the interrupt routine. Several previous coding attempts were unsuccessful because

of the critical sequence of events which must occur for proper operation. Notably, to reset the interrupt level from the MC68488, the interrupt status register must be read and this must be done before the data byte in the data-in register is read. Also, because the MC68488 will automatically complete the data transfer handshake sequence on the IEEE 488 instrument data bus, another data byte may have been received by it, prior to the completion of the interrupt service routine for the previous byte of data. Thus, as soon as the interrupt controller is reset, an interrupt to the processor may occur before the return instruction is executed. It is especially critical when long data streams are received from the IEEE 488 bus in bursts. If special care is not taken to assure that the available memory for the stack is adequate, the results are unpredictable. The compiler will not normally allow enough stack space for this situation unless the interrupt procedure is also labeled REENTRANT. The problem can also be resolved when the code is located by the ISIS-II Locator. The program designer can specify the lower bound for the stack, as well as specifying exactly where the code and data segments are to be located.

Programming procedures to effect the various options available from the MC68488 are not well documented in the references received. However, with a knowledge of the IEEE 488 bus protocol and the functional descriptions of the chip,

the designer will be able to successfully experiment with its operations and achieve an effective and viable interface.

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE GPIAASCIITST
OBJECT MODULE PLACED IN :F1:GPIATS.OBJ
COMPILER INVOKED BY:  PLM80 :F1:GPIATS.PLM INTVECTOR(4,800H) PAGELENGTH(35) PAGE
                      -WIDTH(80)



   1          GPIAASCIITST: DO;

   2    1     DECLARE DCL LITERALLY 'DECLARE';

   3    1     DCL ADDRSWITREG BYTE AT (0FFE4H);

              /* IO ROUTINE */
   4    1     CONO: PROCEDURE(CHAR) EXTERNAL;
   5    2        DCL CHAR BYTE;
   6    2     END CONO;

              /* PROGRAM INTERRUPT CONTROLLER INITIALIZATION */

   7    1     PRO8259: PROCEDURE(I40R8,INTVECTOR) EXTERNAL;
   8    2         DCL I40R8 BYTE,
                     INTVECTOR ADDRESS;
   9    2     END PRO8259;

  10    1     PIC8259$MASK: PROCEDURE(INTMASK) EXTERNAL;
  11    2         DCL INTMASK BYTE;
  12    2     END PIC8259$MASK;

              /* INITIALIZE GPIA PROCEDURES */
  13    1     SETDEVICEADDRESS: PROCEDURE(SPECIALMODE) EXTERNAL;
```

```
14   2          DCL SPECIALMODE BYTE;
15   2        END SETDEVICEADDRESS;

16   1        GPIAINTERSTAT: PROCEDURE BYTE EXTERNAL;
17   2          END GPIAINTERSTAT;

18   1        GPIAINTERMASK: PROCEDURE(MASKBYTE) EXTERNAL;
19   2          DCL MASKBYTE BYTE;
20   2        END GPIAINTERMASK;

21   1        GPIAWRITEAUXCOMM: PROCEDURE(COMMDIN) EXTERNAL;
22   2          DCL COMMDIN BYTE;
23   2        END GPIAWRITEAUXCOMM;

24   1        GPIAADDRESSMODE: PROCEDURE(ADDRSMODE) EXTERNAL;
25   2          DCL ADDRSMODE BYTE;
26   2        END GPIAADDRESSMODE;

27   1        GPIADATAIN: PROCEDURE BYTE EXTERNAL;
28   2        END GPIADATAIN;

29   1        INTER6: PROCEDURE INTERRUPT 6;
30   2          DCL CHAR BYTE;
31   2        DO;
32   3          CHAR = GPIAINTERSTAT;  /*RESET GPIA IRQ PIN */
33   3          CHAR = GPIADATAIN;
34   3          CALL COND(CHAR);
35   3          OUTPUT(0D8H) = 20H;
36   3          ENABLE;
37   3        END;
38   2        END INTER6;
```

```
  39   1       DO;
  40   2           DISABLE;
  41   2           CALL SETDEVICEADDRESS(00);
  42   2           CALL GPIAWRITEAUXCOMM(000);
  43   2           CALL GPIAINTERMASK(1000$0001B);
  44   2           CALL GPIAADDRESSMODE(0000$0000B);
  45   2           CALL PRO8259(4,0800H);
  46   2           CALL PIC8259$MASK(1011$1111B);
  47   2           ENABLE;
  48   2           DO WHILE 1;
  49   3            END;
  50   2       END;

  51   1       END GPIAASCIITST;
```



```
MODULE INFORMATION:

     CODE AREA SIZE     = 004DH     77D
     VARIABLE AREA SIZE = 0001H      1D
     MAXIMUM STACK SIZE = 000AH     10D
     71 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

PROGRAM: HEXMEM

DESCRIPTION: The HEXMEM program loads a hexadecimal formatted file into MDS memory beginning at 8000H. This file is then converted into machine instructions beginning at 9000H. Any displacement of the object code from address 00H will be the displacement of that code from 9000H in MDS memory. An example would be a file normally intended to be loaded at 0800H. When loaded into MDS memory with HEXMEM call, the code will now be at 9800H.

The primary use of this program is to aid EPROM programming with the ICOM PROM Programmer. The PROM Programmer is inserted into the SBC 80/20's systems bus and ICE-80 used to map the memory space of the MDS where the program is loaded into the usable memory range of the SBC 80/20. Example 4 in Appendix D provides a step-by-step listing of the procedures required.

LIMITATIONS: The HEXMEM program can only load from the :F1:HEXMEM.HEX file. This file was created by using the OBJHEX system call from the ISIS-II utility programs. Thus, all code which was to be loaded with the HEXMEM program must be in the :F1:HEXMEM.HEX file.

The base address at which the program begins converting the hexadecimal file is fixed by internal assignment of the pointer variables.

RECOMMENDED MODIFICATIONS: The restriction on the file name could be resolved by allowing the input file name to be entered at the console.

The base address limitation could be removed by requesting the user to input a desired offset or bias.

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE HEXMEM
OBJECT MODULE PLACED IN :F1:HEXMEM.OBJ
COMPILER INVOKED BY:  PLM80 :F1:HEXMEM.PLM PAGELENGTH(35) PAGEWIDTH(80)



   1          HEXMEM: DO;

   2   1      DECLARE DCL LITERALLY 'DECLARE';
   3   1      DECLARE LIT LITERALLY 'LITERALLY';

   4   1          DCL RETURNSTAT ADDRESS;
   5   1          DCL (I,NUMCHARS,NUMBYTE,INBUFFERPTR) ADDRESS;
   6   1          DCL (AFTN,BYTELOAD) ADDRESS;
   7   1          DCL INBUFFER BASED INBUFFERPTR BYTE;
   8   1          DCL (LINEBEGIN,MEMBEGIN) ADDRESS;
   9   1          DCL MEMPTR ADDRESS;
  10   1          DCL MEMBYTE BASED MEMPTR BYTE;
  11   1          DCL PAIRPTR ADDRESS;
  12   1          DCL ASCIIPAIR BASED PAIRPTR ADDRESS;
  13   1          DCL (NUMLINES,J,MEMBIAS) ADDRESS;

  14   1      OPEN:
              PROCEDURE(AFTPTR,FILE,ACCESS,MODE,STATUS) EXTERNAL;
  15   2          DCL (AFTPTR,FILE,ACCESS,MODE,STATUS) ADDRESS;
  16   2          END OPEN;

  17   1      CLOSE:
            PROCEDURE (AFT,STATUS) EXTERNAL;
  18   2          DCL (AFT,STATUS) ADDRESS;
```

```
19   2          END CLOSE;

20   1        READ:
                PROCEDURE (AFT, BUFFER, COUNT, ACTUAL, STATUS) EXTERNAL;
21   2            DCL (AFT, BUFFER, COUNT, ACTUAL, STATUS) ADDRESS;
22   2          END READ;


23   1        WRITE:
                PROCEDURE (AFT, BUFFER, COUNT, STATUS) EXTERNAL;
24   2            DCL (AFT, BUFFER, COUNT, STATUS) ADDRESS;
25   2          END WRITE;

26   1        ERROR:
                PROCEDURE (ERRNUM) EXTERNAL;
27   2            DCL ERRNUM ADDRESS;
28   2          END ERROR;

29   1        EXIT:
                PROCEDURE EXTERNAL;
30   2          END EXIT;

31   1        ASCIITOHEX:
                PROCEDURE(ASCIIPAIR) BYTE;
32   2            DCL ASCIIPAIR ADDRESS;
33   2            DCL (HEXHIGH, HEXLOW) BYTE;

34   2            CONVERT: PROCEDURE(LETTER)BYTE;
35   3              DCL LETTER BYTE;

36   3                IF LETTER > 39H
                       THEN
```

```
37   3                    RETURN LETTER - 37H;
                        ELSE
38   3                    RETURN LETTER - 30H;
39   3              END CONVERT;

               /* SINCE THE MEMORY IS LOADED FROM THE DISK
                  THE VALUES STORED IN MEMORY WOULD BE IN
                  REVERSE ORDER TO WHAT THE NORMAL ARRANGEMENT WOULD BE */

40   2              HEXLOW = HIGH(ASCIIPAIR);
41   2              HEXHIGH = LOW(ASCIIPAIR);
42   2              RETURN (SHL(CONVERT(HEXHIGH),4) + CONVERT(HEXLOW));
43   2         END ASCIITOHEX;

44   1         BYTEFIELD: PROCEDURE BYTE;
45   2             DCL PAIRPTR ADDRESS;
46   2             DCL ASCIIPAIR BASED PAIRPTR ADDRESS;

47   2             PAIRPTR = LINEBEGIN + 1;
48   2             RETURN ASCIITOHEX(ASCIIPAIR);
49   2         END BYTEFIELD;

50   1         ADDRESSFIELD: PROCEDURE ADDRESS;

51   2             DCL PAIRPTR ADDRESS;
52   2             DCL ASCIIPAIR BASED PAIRPTR ADDRESS;
53   2             DCL (HIGHBYTE,LOWBYTE) ADDRESS;

54   2             PAIRPTR = LINEBEGIN + 3;
55   2             HIGHBYTE = DOUBLE(ASCIITOHEX(ASCIIPAIR));

56   2             PAIRPTR = LINEBEGIN + 5;
```

```
 57   2          LOWBYTE = DOUBLE(ASCIITOHEX(ASCIIPAIR));

 58   2          RETURN (SHL(HIGHBYTE,8) + LOWBYTE);
 59   2      END ADDRESSFIELD;



 60   1      CALL OPEN(.AFTN,.('#:F1:HEXMEM.HEX',0DH),1,0,.RETURNSTAT);
 61   1          IF RETURNSTAT > 0 THEN
 62   1                DO;
 63   2                CALL ERROR(RETURNSTAT);
 64   2                CALL EXIT;
 65   2                END;
 66   1      BYTELOAD = 1;
 67   1      INBUFFERPTR = 8000H;
 68   1      DO WHILE BYTELOAD <> 0;
 69   2      CALL READ(AFTN,INBUFFERPTR,1,.BYTELOAD,.RETURNSTAT);
 70   2          INBUFFERPTR = INBUFFERPTR + 1;
 71   2        END;
 72   1      NUMCHARS = INBUFFERPTR - 8000H;
 73   1      INBUFFERPTR = 8000H;
 74   1      NUMLINES = 0;

 75   1      DO I = 1 TO NUMCHARS;
 76   2      CALL WRITE(0,INBUFFERPTR,1,.RETURNSTAT);
 77   2      INBUFFERPTR = INBUFFERPTR + 1;
 78   2        IF INBUFFER = 0DH THEN NUMLINES = NUMLINES + 1;
 80   2      END;

 81   1      CALL CLOSE(AFTN,.RETURNSTAT);

 82   1      LINEBEGIN = 8000H;
```

```
 83   1       MEMBEGIN = 9000H;
 84   1       DO J = 1 TO NUMLINES;
 85   2           MEMPTR = MEMBEGIN + ADDRESSFIELD;

 86   2       IF (NUMBYTE := BYTEFIELD) > 0
                 THEN
 87   2           DO I = 1 TO NUMBYTE;
 88   3               PAIRPTR = LINEBEGIN + 7 + (2*I);
 89   3               MEMBYTE = ASCIITOHEX(ASCIIPAIR);
 90   3               MEMPTR = MEMPTR + 1;
 91   3           END;

 92   2           LINEBEGIN = LINEBEGIN + 13 + (2*NUMBYTE);

 93   2       END;
 94   1       CALL EXIT;

 95   1       END HEXMEM;
```


MODULE INFORMATION:

```
     CODE AREA SIZE     = 0211H    529D
     VARIABLE AREA SIZE = 0029H     41D
     MAXIMUM STACK SIZE = 0008H      8D
     140 LINES READ
     0 PROGRAM ERROR(S)
```

END OF PL/M-80 COMPILATION

APPENDIX C

CONSTRUCTION NOTES

A. GENERAL

This appendix lists hardware modification made to the various components of the DFDR/M. Also contained herein are tables listing the electrical connection for cables and connectors and a schematic of the interface board.

B. SBC 80/20

The following modifications have been made to the SBC 80/20 used in this thesis. These changes were made to meet special requirements of the project.

1. Fail Safe Timer

The fail safe timer input to the ready line was disabled by the removal of the jumper between connector pins 137-138 [Ref. 3]. The removal of the fail safe timer simplifies testing. With this function disabled, the system designer can recognize the failure of one of the components to respond with a ready signal. After the system has been debugged and operational tested, this feature can be reconnected.

2. EIA Interface

Pins eight and ten of the SBC plug J3 have been shorted to provide EIA clear to send to the terminal output [Ref. 19].

3. Interrupt

Connector pins 31 and 36-39 were connected to allow the use of interrupts without utilizing all of interrupt seven's inputs. Interrupts five and six have been implemented by connecting connector pins 21 to 48 and 30 to 49 respectively.

4. MPU Ø2

The MPU Ø2 timing signal to the 68488 GPIA interface circuitry was obtained by connecting pin 6 of A33 to P1/27.

5. 17V Power Supply

The power source for the 17V regulated power supply circuitry on the interface board was obtained by jumping pin 3 of connector P1 on the power supply to connectors J3A/1 and J3A/3 of the Mother Board.

6. Bus Priority

Since the SBC 80/20 is the only SBC used in the system, J5/15 was connected to J3/15 to provide an active low signal whenever the MBM-GPIA interface board is inserted into the Mother Board.

7. System Reset

The front panel reset signal connected to J5A/38 was jumpered to J3A/38 to provide system reset signal to the MBM controller and the 68488 GPIA.

8. Memory

Table I provides a map on SBC 80/20 memory usage.

C. MBM CONTROLLER BOARD

The MBM controller board was implemented as specified by Ref. 6 with the following modifications:

1. Wiring

Connector pin 14 was jumpered to pin 15.

2. Power Failure

Power failure warning circuitry has not been implemented.

3. Addressing and Data

The data and address lines listed in Ref. 6 are implemented differently. Table II indicates the correspondence and the contents of the two address decode PROMs.

4. Redundancy PROM

Table III is the final mask for MBM chip used. The table also shows the contents of the redundancy PROM, U11.

TABLE I

DFDR/M MEMORY MAP

Note: Include ICOM PROM Programmer board with resident monitor, and MBM controller and MC68488 GPIA usage.

ADDRESS RANGE	TYPE	USAGE
0000-06AE	EPROM	SBC 80/20 Monitor
06AF-07FF	EPROM	NOT USED
0800-0BFF	EPROM	MC68488 GPIA Test Program
0C00-0FFF	EPROM	NOT USED
1000-2FFF	Not Implememted	
3000-3FFF	RAM	Available to user
Note:	Space used by various programs	
	3C20-3D15	ICOM PROM Programmer
	3000-3020	68488 GPIA Test Program
	3F81-3FFF	SBC 80/20 Monitor
4000-BFFF	Not Implemented	
C000-DBFF	EPROM	Not used, only available if ICOM PROM programmer is installed.
DC00-DFFF	EPROM	ICOM PROM Programmer Monitor if installed.
E000-FFDF	Not Implemented	
FFE0-FFE7		MC68488 GPIA mapped to this area.
FFE8-FFEF	Not Implemented	
FFF0-FFFE		MBM controller mapped to this area.
FFFF	Not Implemented	

TABLE II

ADDRESS AND DATA LINE CORRESPONDENCE

SCHEMATIC LINE CALLOUT	LINE CALLOUT
A14	ADR0
A13	ADR1
A12	ADR2
A11	ADR3
A2	ADR4
A1	ADR5
A0	ADR6
A10	ADR7
A9	ADR8
A8	ADR9
A7	ADRA
A6	ADRB
A5	ADRC
A4	ADRD
A3	ADRE*
D7	DAT0
D6	DAT1
D5	DAT2
D4	DAT3
D3	DAT4
D2	DAT5
D1	DAT6
D0	DAT7

U1 PROM Contents

Address Range	Value
00-06,08-0E,10-FF	B
07	E
0F	8

U2 PROM Contents

Address Range	Value
00-FE	F
FF	0

TABLE III

MAGNETIC BUBBLE FINAL MASK AND
REDUNDANCY PROM CONTENTS

FINAL MASK

MODULE 89-92-10

00E0 0042 01C0 0810 0000 0000 0000 0000 0000 0007

BAD LOOP ADDRESSED (HEXIDECIMAL)

08, 09, 0A, 19, 1E, 27, 28, 29, 34, 3B

REDUNDANCY PROM CONTENTS (U11)

Address Range	Value
00-07	F
08-0A	E
0B-18	F
19	E
1A-1D	F
1E	E
1F-26	F
27-29	E
2A-33	F
34	E
35-3A	F
3B	E
3C-FF	F

TABLE IV

CONTROLLER/BUBBLE CAGE BACKPLANE

MBM BOARD PLUG P1 PIN"	FUNCTION	CONTROLLER BOARD PLUG P1 PIN"	INTERFACE BOARD Plug PE PIN"
1,A	GROUND	1	1
2,21,B,Y	+5 V	21	5
3,11,12,13	+12 VDC	3	4
C,M,N,	+12 VDC	3	4
4,D,P	+17 VDC	D	EXTERNAL SUPPLY
5	DAT OUT	5	
6	XOUT/	6	
7-8	UNUSED	---	---
9	ANN/	9	
10	UNUSED	---	---
14-15	UNUSED	---	---
16	CYA/	16	
17	XIN/	14	
18	CXB/	17	
19	CLAMP/	11	
20	STROBE/	13	
22,Z	-5 VDC	22	2&3
E	BDEN/	1	
F-J	UNUSED	---	---
K	REP/	K	
L	UNUSED	---	---
R	UNUSED	---	---
S	CYB/	S	
T	GEN/	12	
U	CXA/	T	
V-X	UNUSED	---	---

TABLE V

CABLE-INTERFACE BOARD TO CONTROLLER

INTERFACE BOARD PLUG P-2/CABLE SOCKET J2-A			CONTROLLER PLUG P2 CABLE SOCKET J2-B
PIN "	FUNCTION	COLOR	PIN "
A-1	SIGNAL GROUND	BROWN	1
A-2	ADDRESS (B)	RED	2
A-3	ADDRESS (C)	ORANGE	3
A-4	BOARD SELECT (A)	YELLOW	4
A-5	ADDRESS (D)	GREEN	5
A-6	ADDRESS (6)	BLUE	6
A-7	ADDRESS (A)	VIOLET	7
A-8	ADDRESS (5)	GREY	8
A-9	ADDRESS (7)	WHITE	9
A-10	ADDRESS (4)	BLACK	10
A-11	ADDRESS (8)	BROWN	11
A-12	DBIN	RED	12
A-13	ADDRESS (9)	ORANGE	13
A-14	MEMEN	YELLOW	14
A-13	ADDRESS (E)	GREEN	15
A-16	BOARD SELECT (C)	BLUE	16
A-17	HARDWARE RESET	VIOLET	17
A-18	BOARD SELECT (B)	GREY	18
A-19	POWER BAD	WHITE	19
A-20	----------	BLACK	20
A-21	ADDRESS (0)	BROWN	21
A-22	----------	RED	22
A-23	ADDRESS (1)	ORANGE	23
A-24	----------	YELLOW	24
A-25	ADDRESS (2)	GREEN	25
A-26	----------	BLUE	26
A-27	ADDRESS (3)	VIOLET	27
A-28	DATA (0)	GREY	28
A-29	CLOCK (Ø2)	WHITE	29

A-30	DATA (1)	BLACK	30
A-31	READY TO MICROPROCESSOR	BROWN	31
A-32	DATA (2)	RED	32
A-33	----------	ORANGE	33
A-34	DATA (3)	YELLOW	34
A-35	DATA (7)	GREEN	35
A-36	DATA (4)	BLUE	36
A-37	----------	VIOLET	37
A-38	DATA (5)	GREY	38
A-39	INTERRUPT TO CPU	WHITE	39
A-40	DATA (6)	BLACK	40

CONNECTIONS FROM J2-A/B TO CARDCAGE BACK PLANE

			MBM BOARD PIN "
B-]	GROUND	BLACK	1
B-2	+5 VDC	RED	22
B-3	+5 VDC	RED	22
B-4	+12 VDC	WHITE	12
B-5	-5 VDC	GREEN	2
---	UNUSED	BLUE	---
EXTERNAL SUPPLY	+17 V	BROWN(PURPLE)	4

NOTE: J2-A IS A 100 PIN EDGE CARD CONNECTION.
J2-B IS A 40 PIN A-D PRODUCTS CONNECTOR.

TABLE VI

DIRECTORY OF DISKETTE THESIS.1

GENERAL:

This diskette was used primarily as a systems diskette. It contains all the ISIS-II utilities as well as the ISIS-II 8080/8085 Macro Assembler and the software for use with the In Circuit Emulator/80. Only those files other than the ISIS-II utilities will be listed.

FILE NAME	DESCRIPTION
ASM80	ISIS-II 8080/8085 Macro Assembler
ICE80	In Circuit Emulator/80 Software
PRIM.PLM	Source code for module PRIMTIVE which is used in the library AIRDAT.LIB. This is a back-up file for code on diskette THESIS.A
GPIA.PLM	Source code for module GPIAMOD which is used in the library AIRDAT.LIB. This is a back-up for code on diskette THESIS.A

TABLE VII

DIRECTORY OF DISKETTE THESIS.2

GENERAL:

This diskette was used primarily as a system diskette. It contains all of the ISIS-II utilities as well as the ISIS-II PL/M-80 Compiler and several special executable files which are described.

FILE NAME	DESCRIPTION
PLM80	ISIS-II PL/M-80 Compiler
HEXMEM	Executable object file for the program HEXMEM.PLM. A source code listing is found in Appendix B. When executed, this program loads a hexadecimal formatted file into MDS memory for subsequent programming into a PROM using ICE80.
MARLST	Executable object file for the program MARLST.PLM. The program was written to adjust the left margin of the LST file created by the PL/M-80 Compiler while printing the file to the lineprinter.
AIRDAT.LIB	A library file created to contain previously compiled modules of code for the DFDR/M. The ISIS-II System Library (SYSTEM.LIB) and the PL/M-80 Library (PLM80.LIB) are included to simplify linking of other modules.

TABLE VIII

DIRECTORY OF DISKETTE THESIS.A

GENERAL:

This diskette was used primarily for the storage of programs written for DFDR/M interface testing of both the MBM and the MC68488 GPIA. Five of the programs are listed in Appendix B and will not be described here. Each file will have an associated object file (.OBJ) and list file (.LST). Main program files will also have a linked relocatable object file (.LNK) and the resulting absolute executable file (file name with no attribute). These files will not be listed in this abbreviated directory.

FILE NAME	DESCRIPTION
RDWR.PLM	Source code and description in Appendix B.
GPIA.PLM	Source code and description in Appendix B.
GPIATS.PLM	Source code and description in Appendix B.
AIRDAT.LIB	This is a back-up file for the file on diskette THESIS.2. See Table VII for description.
PRIM. PLM	Source code and description in Appendix B.
HEXMEM.PLM	Source code and description in Appendix B.
FIFOTS.PLM	This program written to verify to interface between the SBC 80/20 and the MBM controller. Data is written to the controller's FIFO and read back

	and compared with the value written validate the interface. No attempt is made to write the data to the MBM.
NWRDWR.PLM	This program is identical to RDWR.PLM except for time delays inserted to test if the timing of commands to the MBM controller had any effect on the transfer of data to and from the MBM.

TABLE IX

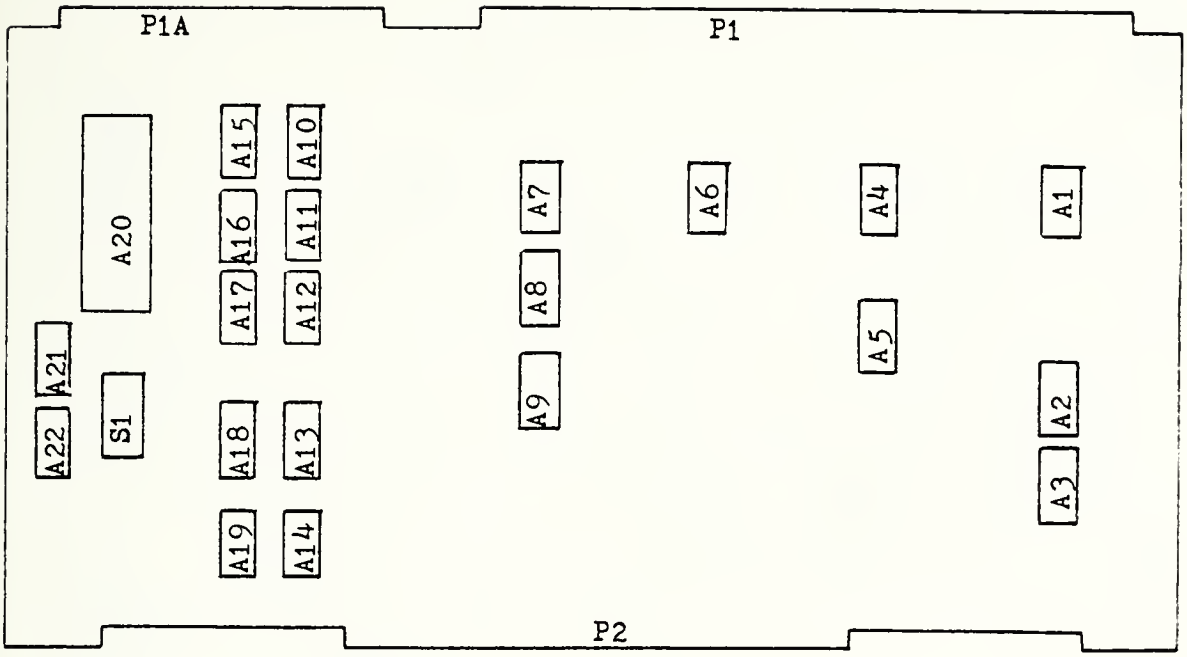
DIRECTORY OF DISKETTE THESIS.C

GENERAL:

This diskette was used primarily for the storage of programs written for testing the operation of the MBM. Each program is a main program module and has associated with it a relocatable object file (.OBJ), a list file (.LST), a linked relocatable object file (.LNK), and the resulting absolute executable file (file name with no attribute). These files are not listed in this abbreviated directory.

FILE NAME	DESCRIPTION
BUBWRT.PLM	This program was used to write sequential bit patterns to the MBM while monitoring the generate signal from the MBM controller with an oscilloscope to verify its presence and timing.
BUBINT.PLM	This program was used to test the interrupt feature of the MBM controller. The program worked when emulation was done at less than real-time. The problem may be due to improper timing of the write page command to the MBM.
BUBPAG.PLM	This program was used to test the MBM module. Between a write of a page of data to the MBM, the FIFO is zeroed to verify that the data in the FIFO was being changed when a read was executed.

BXINTS.PLM	This program was used to continuously write a page of data to a specific page to allow the Transfer in and Transfer out signals to be monitored with an oscilloscope.
MARLST.PLM	This program was used to print the list file created by the compiler to the line printer with an adjusted left margin.
BL1.PLM	This program was used to verify the transfer of a page of data from the FIFO of the MBM controller to the MBM. Between the write of the page and a subsequent read, the major loop of the MBM is cleared by a MBM controller initialize command.



LOCATION	IC TYPE NUMBER	DESCRIPTION
A1	7400	QUAD 2-INPUT NAND GATE
A2	7474	DUAL D EDGE-TRIGGERED FLIP-FLOP
A3	7474	DUAL D EDGE-TRIGGERED FLIP-FLOP
A4	7404	HEX INVERTER
A5	7404	HEX INVERTER
A6	7404	HEX INVERTER
A7	7485	4-BIT MAGNITUDE COMPARITOR
A8	7485	4-BIT MAGNITUDE COMPARITOR
A9	7485	4-BIT MAGNITUDE COMPARITOR
A10	74126	QUAD TRI-STATE DRIVER(HIGH ENABLE)
A11	74125	QUAD TRI-STATE DRIVER(LOW ENABLE)
A12	7407	HEX DRIVER(OPEN COLLECTOR) NON-INVERTING
A13	74125	QUAD TRI-STATE DRIVER(LOW ENABLE)
A14	7407	HEX DRIVER(OPEN COLLECTOR)NON-INVERTING
A15	74126	QUAD TRI-STATE DRIVER(HIGH ENABLE)
A16	74125	QUAD TRI-STATE DRIVER(LOW ENABLE)
A17	7407	HEX DRIVER(OPEN COLLECTOR)NON-INVERTING
A18	74126	QUAD TRI-STATE DRIVER(HIGH ENABLE)
A19	7404	HEX INVERTER
A20	MC68488P	GENERAL PURPOSE INTERFACE ADAPTER
A21	74125	QUAD TRI-STATE DRIVER(LOW ENABLE)
A22	74125	QUAD TRI-STATE DRIVER(LOW ENABLE)
S1		7-BIT DIP SWITCH

INTERFACE BOARD SHCEMATIC
Figure 20 (Sheet 1 of 6)

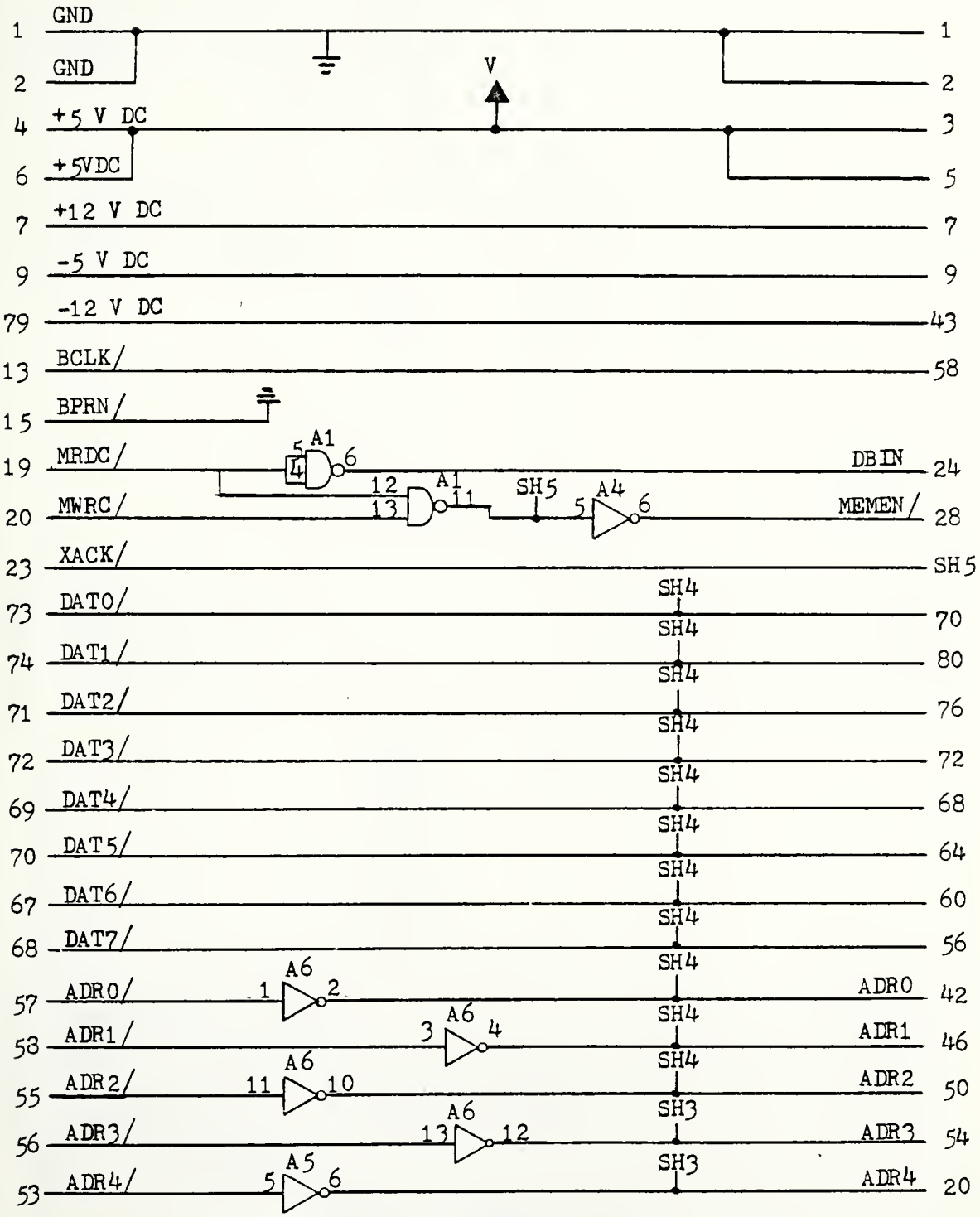


Figure 20 (Sheet 2 of 6)

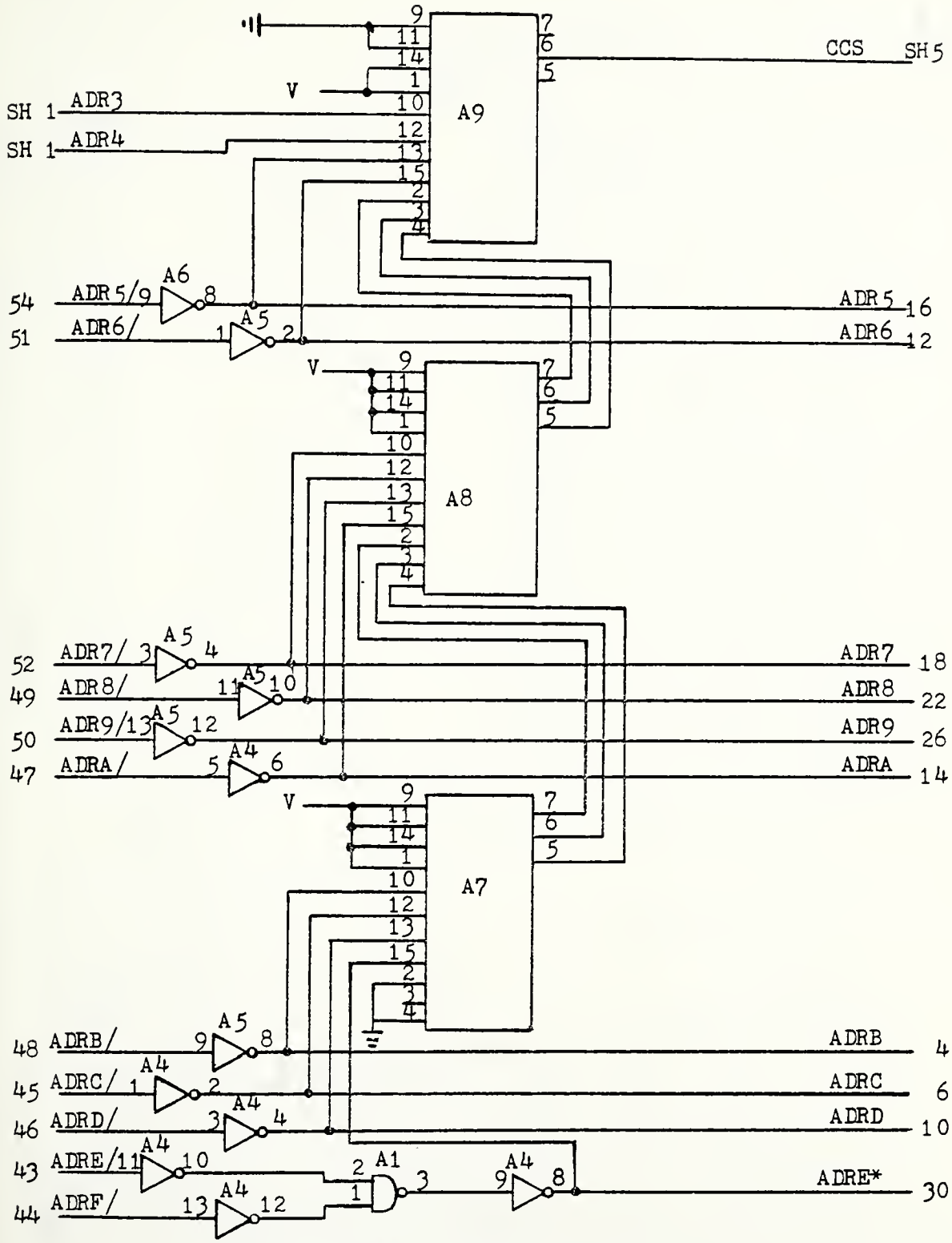


Figure 20 (Sheet 3 of 6)

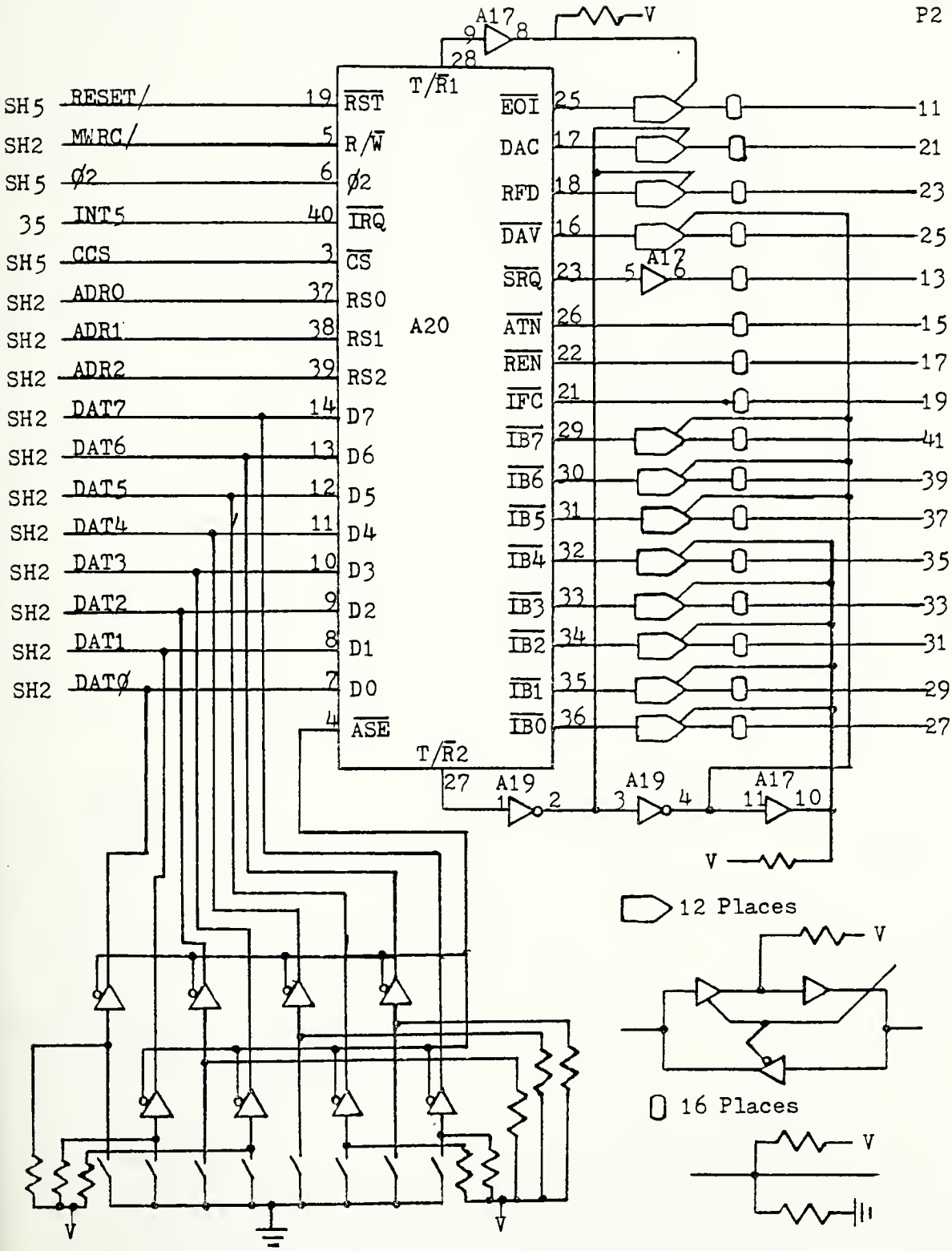


Figure 20 (Sheet 4 of 6)

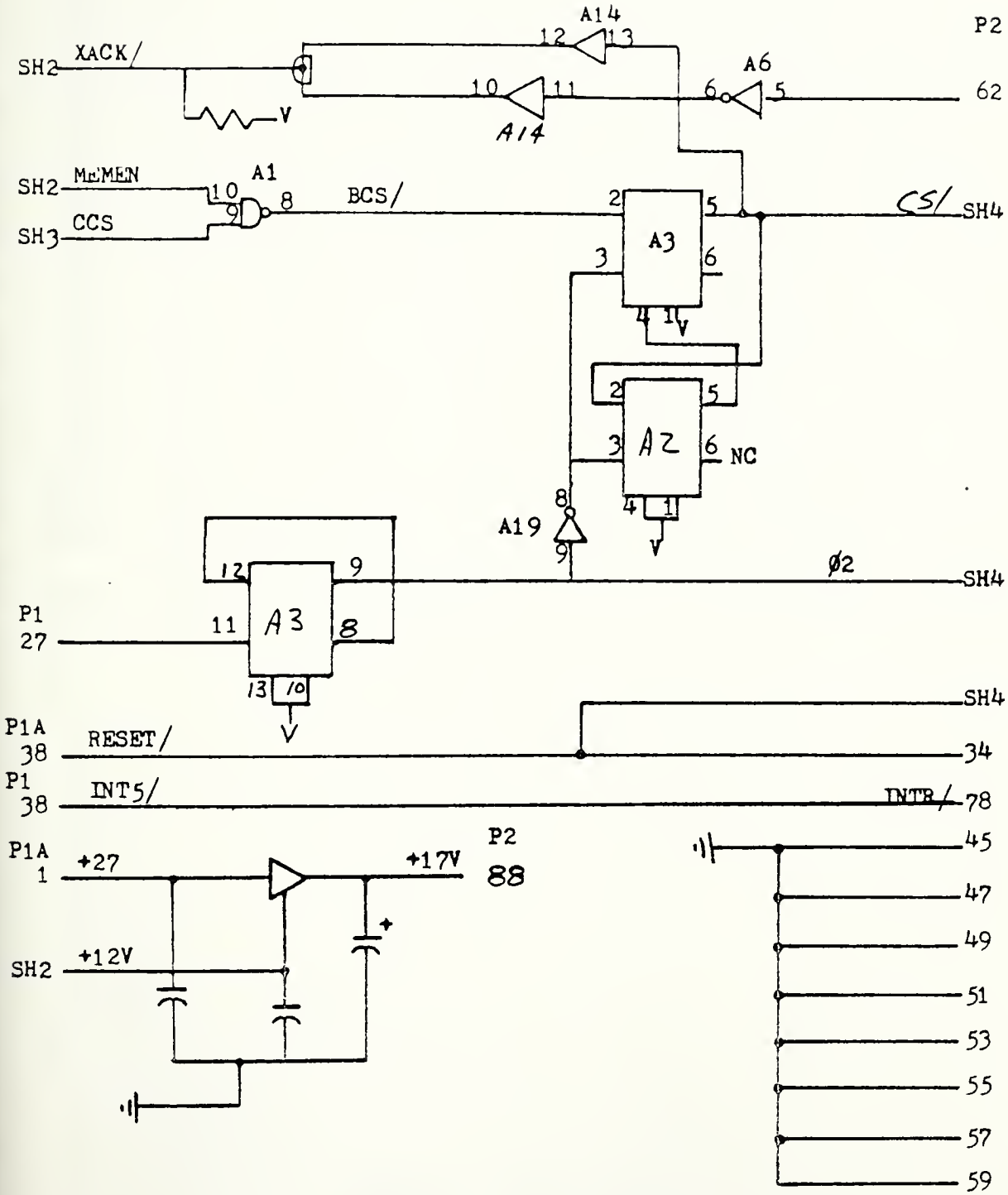


Figure 20 (Sheet 5 of 6)

SIGNAL	74126 LOCATION	74126 PIN IN	74126 PIN OUT	74126 CONTROL	74125 LOCATION	74125 PIN IN	74125 PIN OUT	74125 CONTROL	7404 LOCATION	7404 PIN IN	7404 PIN OUT
EOI	A18	2	3	1	A13	2	3	1	A14	1	2
DAC	A18	5	6	4	A13	5	6	4	A14	3	4
RFD	A18	12	11	13	A13	12	11	13	A14	5	6
DAV	A18	9	8	10	A13	9	8	10	A14	9	8
IB7	A10	2	3	1	A11	2	3	1	A12	1	2
IB6	A10	5	6	4	A11	5	6	4	A12	3	4
IB5	A10	9	8	10	A11	9	8	10	A12	5	6
IB4	A10	12	11	13	A11	12	11	13	A12	13	12
IB3	A15	2	3	1	A16	2	3	1	A12	11	10
IB2	A15	5	6	4	A16	5	6	4	A12	9	8
IB1	A15	9	8	10	A16	9	8	10	A17	1	2
IB0	A15	12	11	13	A16	12	11	13	A17	3	4

Figure 20 (Sheet 6 of 6)

APPENDIX D

SYSTEM OPERATING EXAMPLES

This appendix is included to provide a listing of sequential commands used to accomplish specific tasks for development and testing of the hardware and software components. Six examples are provided here. The first and second examples will specify the mechanical steps required to initialize the MDS system and the SBC 80/20, respectively. The third example demonstrates the development of a software program and subsequent execution in the SBC 80/20 using ICE-80. The fourth example shows the steps taken to change the source file of a module in the library file created to simplify program development. The library file is updated and an executable module is created, linked and located. ICE-80 is then used to verify program correctness.

The fifth example will provide a step-by-step demonstration on the method used to place a program written for the SBC 80/20 into an EPROM for subsequent execution without the emulator.

The last example demonstrates the use of the SBC 80/20 and the HP-9825A calculator to verify the operation of the MC68488 interface between the SBC 80/20 and the HP-IB bus (identical to the bus specified in the IEEE 488 standard).

The symbols used in the examples are defined as follows:

- Prompt symbol displayed by the ISIS-II operating system.

*	Prompt symbol displayed by the ISIS-II utilities (Text Editor, Library Manager, ICE-80, etc.).
**	Prompt symbol displayed by the ISIS-II operating system when '&' was used for command line continuation.
.	Prompt symbol displayed by the SBC 80/20's Monitor.
%	Prompt symbol displayed by the ICOM PROM Programmer.
$	The symbol displayed by the Text Editor when the escape (ESC) key on the input terminal is depressed.
()	All comments used to explain the sequence of commands are enclosed in parentheses.

EXAMPLE 1:

Purpose:

Demonstrate MDS start-up procedures as described in Ref. 12.

Hardware Configuration:

MDS system with dual diskette unit and terminal.

Steps:

1. Switch on all three units.
2. Insert a systems diskette into drive 0.
3. Depress upper half of B00T switch on the MDS front panel.
4. Depress upper half of RESET switch and release.
5. Depress space bar on the I/O terminal.
6. Depress bottom half of B00T switch.

(ISIS sign-on message should appear on the I/O terminal. Note: The monitor module has been modified to allow switch selection of 300 or 2400 BAUD rate for I/O interfacing. 300 BAUD, switch one is on with all others off. 2400 BAUD, switch five is on with all others off [Ref. 3, pps. 86-87].)

EXAMPLE 2

Purpose:

Demonstrate the initialization of the SBC 80/20 Monitor.

Hardware Configuration:

SBC 80/20 with 8080 CPU

Silent 700 terminal

(Note: Initialization of the SBC 80/20 is essentially identical if ICE-80 is used in place of the 8080 CPU. An example of ICE-80 operation is covered later.)

Steps:

1. Switch on the SBC 80/20 and Silent 700.

2. Momentarily depress the RESET switch on the SBC 80/20 front panel.

3. Strike the 'U' key on the terminal six times. (SBC 80/20 sign-on message should appear on the terminal unit. Note: The SBC 80/20 utilizes automatic BAUD rate selection. Thus, other terminal units can be utilized.)

EXAMPLE 3

Purpose:

Demonstrate software development on the MDS system with subsequent testing and debugging using ICE-80.

Hardware Configuration:

MDS with ICE-80 hardware.

SBC 80/20 with ICE-80 Emulation Cord installed in place of 8080 CPU.

TI Silent 700 for SBC 80/20 terminal unit.

Interface Board for MBM operation.

Both MBM boards inserted in the MBM card cage.

Steps:

(Place diskette THESIS.2 in drive 0. Place diskette THESIS.A in drive 1. Initialize MDS system as in Example 1.)

```
ISIS-II, V6.2
-EDIT :F1:RDWR.PLM
ISIS-II  TEXT EDITOR, V1.6
*    (Enter text as required.  See Reference 12 for text
     editing instructions.  After completion of the editing
     process, the source file is compiled using the PL/M 80
     compiler.)
*E$$
-PLM80  :F1:RDWR.PLM
PL/M-80 COMPILER, V3.0
PL/M-80 COMPILATION COMPLETE      2 ERROR(S)
```

```
    (Any syntax errors that were present in the file are
    documented in the list file created by the compiler.
    If only a few errors are present, a quick look at the
    file can be obtained by using the COPY command.)

-COPY :F1:RDWR.LST TO :C0:

    (However, if the number of errors is significant, the
    EDIT command can be used to examine the file line by
    line.)

-EDIT :F1:RDWR.LST

ISIS-II  TEXT EDITOR, V1.6

    (After the errors are noted, correction to the source
    file is made using the editor.)

*E$$

EDIT  :F1:RDWR.PLM

ISIS-II  TEXT EDITOR, V1.6

*
    (Make the necessary corrections)

*E$$

-PLM80  :F1:RDWR.PLM

PL/M-80 COMPILER, V3.0

PL/M-80 COMPILATION COMPLETE     0 ERROR(S)

    (Programs which use procedures in the AIRDAT.LIB library
    file must be linked to it.  The MDS System Library
    (SYSTEM.LIB) and the PL/M80 Library (PLM80.LIB) are
    included in the AIRDAT.LIB file.)

-LINK  :F1:RDWR.OBJ,AIRDAT.LIB TO :F1:RDWR.LNK

    (The relocatable object module produced by the linkage
    editor must be located for use in the SBC 80/20.)

-LOCATE :F1:RDWR.LNK  CODE (3000H) MAP

MEMORY MAP OF MODULE RDWR
READ FROM FILE :F1:RDWR.LNK
WRITTEN TO FILE :F1:RDWR
START ADDRESS 3040H
```

```
 ADDRESS    LENGTH    SEGMENT
   3000H      0430H     CODE
   3430H      0014H     STACK
   3444H      0028H     DATA
   346CH      C254H     MEMORY
```

(ICE-80 can now be used to place the object code into the SBC 80/20's RAM space. Turn on the SBC 80/20 and its terminal unit. Replace diskette in drive 0 with the diskette labeled THESIS.1.)

```
-ICE80

ISIS-II  ICE-80,V4.0
```

(On ICE-80 initialization, memory and I/O ports are guarded. The transform command is used to unguard or map the SBC 80/20 memory. The following commands will unguard all addressable memory. If the executing program attempts to access non-existent memory, a time-out error will terminate the emulation.)

```
*XFORM MEMORY 0 TO 15 UNGUARDED
```

(The above command line can be reduced to XF MEM 0 TO 15 UNG.)

```
*XF IO 0 TO 15 UNG
```

(The next command allows all display of memory or registers contents to be shown in hexadecimal.)

```
*BASE HEX
```

(The object file can now be loaded into SBC 80/20 RAM.)

```
*LOAD :F1:RDWR
```

(Emulation can now begin, the number of options available to the user are too numerous to list. This example requires that the SBC 80/20's USART for terminal communication be initialized using the SBC 80/20's Monitor.)

```
*GO FROM 0 UNTIL 3040H EXECUTED

EMULATION BEGUN
```

(The SBC 80/20 can now be initialized as in Example 2. The SBC 80/20's Monitor can be used for program execution, however, ICE-80 has a large selection of commands for execution control. By using the Monitor's GO command, .G3040, control is returned to ICE-80.)

```
EMULATION TERMINATED AT 3040H
```

(The following command allows for a group of program instructions to be executed with a dump of register contents after each instruction.)

```
*STEP BY 1 INSTRUCTION FROM 3040H THEN DUMP CONTINUE COUNT 10
```

(The above command could be used just to monitor the program periodically. The number of instructions executed before the dump can be increased and the specific number of times can be generalized to forever.)

```
*ST BY 10 INST FR 3040H THEN DUMP CONT FOREVER
```

(Emulation can be terminated at any time by using the Interrupt 4 switch on the front panel of the MDS unit.)

```
EMULATION TERMINATED AT 3096H
```

(Memory contents can be displayed and RAM contents changed with the DISPLAY and CHANGE commands, respectively.)

```
*DIS MEM 3000H TO 30FFH
```

(The contents of the memory locations 3000H to 30FFH are displayed.)

```
*CH MEM 3051H=C3H,23H,30H
```

(The contents of adjacent locations starting at 3015H is changed to C3H,23H,30H. Often it is convenient to save this patched program for later execution. The memory contents can be transferred to a diskette file.)

```
*SAVE :F1:RDWR.PAT 3000H TO 34CFH
```

(Control can be returned to ISIS-II with the ICE-80 EXIT command.)

```
*EXIT
```

(To shut down the system, remove the diskettes, shut off all devices.)

EXAMPLE 4

Purpose:

To demonstrate how to update the library file should any of the procedures contained therein require modification. The example will update library AIRDAT.LIB with a new version of module PRIMITIVE. Source code for the module is in PRIM.PLM.

Hardware Configuration:

Only the basic MDS system is used.

Steps:

(Place diskette THESIS.2 in drive 0. Place diskette with the module to be updated in drive 1. Initialize the MDS system as in example 1.)

```
ISIS-II, V6.0

-EDIT  :F1:PRIM.PLM

ISIS-II TEXT EDITOR, V1.6

*
```

(The necessary changes are made to the source code.)

```
*E$$

-PLM80 :F1:PRIM.PLM

PL/M-80 COMPILER, V3.0

PL/M-80 COMPILATION COMPLETE    0 PROGRAM ERROR(S)
```

(Modification of the library file requires the use of the ISIS-II library manager.)

```
-LIB AIRDAT.LIB

ISIS-II LIBRARY MANAGER, V1.0
```

(Delete the old version of module PRIMITIVE from the library.)

```
*DELETE AIRDAT.LIB(PRIMITIVE)
```

(Add the new version of object code for module PRIMITIVE from file PRIM.OBJ.)

```
*ADD :F1:PRIM.OBJ TO AIRDAT.LIB
```

(To return to ISIS-II operating system use the EXIT command.)

```
*EXIT
```

(Be sure to relink any files which require the new version of the library. The following steps show the updating of the program RDWR. It is assumed that the relocatable object module was saved.)

```
-LINK :F1:RDWR.OBJ,AIRDAT.LIB TO :F1:RDWR.LNK

-LOCATE :F1:RDWR.LNK CODE (3000H)
```

(If verification of the program is required, use of ICE-80 will be required. Turn on the SBC 80/20.)

```
-ICE80

ISIS-II ICE-80, V4.0

*XF MEM 0 TO 15 UNG

*XF IO 0 TO 15 UNG
```

(All memory at I/O ports are unguarded. Any reference to unavailable portions will result in time-out termination.)

```
*B H

LOAD :F1:RDWR

*GO FR 0
```

(Initialize the SBC 80/20 as in Example 2. This is done mainly to initialize the SBC 80/20's USART for I/O with the console. If your program contains the USART initialization routine, the GO command could specify the program start address.

There are two methods to determine the start address. One method is to examine the loaded code. The first instruction is normally a load stack pointer immediate. This will normally be near the address specified in the LOCATE command. Constant data such as strings of characters are normally located before the first instruction.

The second method is to use the MAP control with the LOCATE command. The starting address will be listed on the console at the completion of the command.)

EXAMPLE 5

Purpose:

This example provides a step-by-step set of instructions to place a program written for the SBC 80/20 into PROM using ICE-80 for the transfer of code from MDS memory.

Hardware Configuration:

SBC 80/20 with ICE-80 Emulation Cord installed in place of the 8080 CPU.

ICOM PROM Programmer with EPROM in programming socket.

TI Silent 700 used for SBC 80/20 terminal unit.

Interface Board is removed from SBC 80/20 card cage.

Steps:

(Start by loading diskette labeled THESIS.2 in drive 0 and diskette labeled THESIS.A in drive 1. Next edit the file, in this example the file name will be GPIATS.PLM and the file will be placed on the diskette in drive 1.)

```
-EDIT  :F1:GPIATS.PLM

ISIS-II TEXT EDITOR, V1.6
```

(See Reference 12 for use of editor. The file created included an interrupt procedure with the intention of placing the interrupt vector for the SBC 80/20 interrupt controller at 0800H.)

```
*E$$

-PLM80 :F1:GPIATS.PLM INTVECTOR(4,0800H)

ISIS-II PL/M-80 COMPILER, V3.0

PL/M-80 COMPILATION COMPLETE     0 PROGRAM ERROR(S)
```

```
    (The GPIATS.PLM program contains procedures which were
    declared as external.  It requires linking to the
    library module of the DFDR/M and the System library.)

-LINK :F1:GPIATS.OJB,AIRDAT.LIB TO :F1:GPIATS.LNK

    (The compiler reserved space for the interrupt vector
    at 0800H.  The beginning address of code will be
    0820H, all data items must begin at 3000H for SBC
    80/20 RAM.)

-LOCATE :F1:GPIATS.LNK ORDER(CODE,DATA,STACK,MEMORY) CODE
    (0820H) &
**DATA(3000H) STACK(3C00H)

    (The absolute object file created by the locator
    requires conversion to hexadecimal format for use of
    the HEXMEM program for loading of the code into MDS
    memory.  The HEXMEM program requires that the hexi-
    decimal formatted file be placed in a file named
    HEXMEM.HEX on the diskette in drive 1.)

-OBJHEX :F1:GPIATS TO :F1:HEXMEM.HEX

-HEXMEM

    (Replace the diskette in drive 0 with the diskette
    labeled THESIS.1.  Turn on both the SBC 80/20 and
    the Silent 700 terminal unit.)

-ICE80

ISIS-II ICE-80,V4.0

    (In this execution of ICE-80, the 4K RAM area of the
    SBC 80/20 is mapped in the MDS RAM beginning at 9000H.)

*XF MEM o TO 2 UNG

*XF MEM 3 INTO 9

*XF MEM 4 TO 15 UNG

*XF IO 0 TO 15 UNG

*GO FROM 0

    (Subsequent action will be on the terminal unit of
    the SBC 80/20.  Initialize the system as in Example 2.)

SBC 80/20 MONITOR, V1.0

    (Transfer control to the ICOM PROM Programmer.)
```

```
.GDC03

%P3800,3BFF,0

PWR ON $ PRESS KEY

    (PROM programming takes several minutes.)

PWR OFF & PRESS KEY

    (The PROM Programming is complete.  Note:  Before
    shutting down the MDS system, remove the diskettes.)
```

EXAMPLE 6

Purpose:

To demonstrate the use of the GPIATS program to test the MC68488 GPIA.

Hardware Configuration:

SBC 80/20 with 8080 CPU, Interface Board, and TI Silent 700.

HP-9825A calculator with HP-IB capability.

HP-IB connected to socket of Interface Board connector.

Magnetic Bubble Memory boards need not be installed in the MBM card cage.

EPROM that is programmed with the GPIATS program inserted in the third PROM socket of the SBC 80/20.

Steps:

(Initialize the SBC 80/20 as in Example 2. Turn on the HP-9825A calculator.)

```
SBC 80/20 MONITOR, V1.0

.G0820
```

(SBC 80/20 GO command to begin execution of the program. The HP-9825A can now be used to send ASCII string to the Silent 700 through the IEEE 488 interface circuitry. The 'wrt' statement is used on the HP-9825A to transmit the string.)

```
wrt 700,'THIS IS A TEST STRING'
```

(When testing is done, turn off all equipment.)

BIBLIOGRAPHY

1. Baetz, L. N., Study and Design of Flight Data Recording System for Military Aircraft, Master of Science Thesis, Naval Postgraduate School, June 1976.

2. Albertolli, W. R., Microcomputer Based Solid State Crash Data Recorder for Military Aircraft, Master of Science Thesis, Naval Postgraduate School, December 1976.

3. Kane, D. L., Design and Construction of a Flight Monitor and Data Recorder, Master of Science Thesis, Naval Postgraduate School, December 1977.

4. Texas Instruments, Inc., Magnetic Bubble Memories and System Interface Circuits, February 1977.

5. Texas Instruments, Inc., TMS 9916 Bubble Memory Controller, April 1977.

6. Texas Instruments, Inc., Logic Diagram, Bubble Memory Controller, Drawing No. SK326758, May 1977.

7. Texas Instruments, Inc., Electronic Schematic Diagram, BK10103 MAG, Bubble Mem, Printed Ckt Evaluation Board, Drawing No. SK312763, May 1977.

8. The Institute of Electrical and Electronic Engineers, Inc., IEEE Standard Digital Interface for Programmable Instrumentation, IEEE Std 488-1975, December 1974.

9. Motorola Semiconductor Products, Inc., Getting Aboard the 488-1975 Bus, Implementation of the IEEE 488-1975 Instrument Bus with the MC68488 Interface Adapter, October 1977.

10. Motorola Semiconductor Products, Inc., MC68488 General Purpose Interface Adapter, Advance Information, October 1977.

11. Intel Corp., MDS-800 Intellec MDS, Microcomputer Development System, Operator's Manual, Document No. 98-129A, 1976.

12. Intel Corp., ISIS-II System User's Guide, Document No. 98-3068, 1976.

13. Intel Corp., ISIS-II PL/M-80 Compiler, Operator's Manual, Document No. 98-300B, 1977.

14. Intel Corp., ISIS-II 8080/8085 Macro Assembler, Operator's Manual, Document No. 98-292A, 1977

15. Intel Corp., In Circuit Emulator/80, ICE-80 Operator's Manual, Document No. 98-185C, 1976.

16. Intel Corp., In Circuit Emulator/80, Hardware Reference Manual, Document No. 98-167, 1975.

17. Intel Corp., PL/M-80 Programming Manual, Document No. 98-268, 1976.

18. Intel Corp., 8080/8085 Assembly Language Programming Manual, Document No. 98-301A, 1977.

19. Intel Corp., SBC 80/20 and SBC 80/20-4 Single Board Computer Hardware Reference Manual, Document No. 98-317C, 1977.

20. Lancaster, D., TTL Cookbook, Howard W. Sams and Company, Inc., 1974.

21. Intel Corp., MCS-80 User's Manual, Document No. 98-153D, October 1977.

22. ICOM, Model PP80MDS/SBC 80/20-R PROM Programmer Operator's Manual and Software Listing, August 1977.

23. Tektronix, Inc., 464 Oscilloscope Operator's Manual, May 1976.

24. National Transportation Safety Board Letter 2271-C to Federal Aviation Administration, Subject: Safety Recommendations A-78-27 through 29, April 1978.

25. Hewlett-Packard Calculator Products Division, Hewlett-Packard 9825A Calculator Operating and Programming, December 1976.

26. Hewlett-Packard Calculator Products Division, Hewlett-Packard 9825A Calculator General I/O Programming, January 1977.

27. U.S. Army Research and Technology Report TR-51, Preliminary Design of An Accident Information Retrieval System (AIRS), by H. R. Ask, and others, August 1977.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3.	Department Chairman, Code 52 Computer Science Department Naval Postgraduate School Monterey, California 93940	1
4.	Assoc. Professor U. R. Kodres, Code 52Kr Computer Science Department Naval Postgraduate School Monterey, California 93940	1
5.	Capt. Darl E. Easton, USAF Det AK, Electronics Systems Division Langley AFB, Virginia 23665	1
6.	Air Force Institute of Technology/CIDD Wright Patterson AFB, Dayton, Ohio 45433	1
7.	LCDR Frank Burkhead, Code 52BG Computer Science Department Naval Postgraduate School Monterey, California 93940	1

Thesis 176926
E1427 Easton
c.1 Microcomputer based
flight data recorder/
monitor with solid
state memory.